AMIGA
grafica 3D e animazione

Axel Plenge

- RAY-TRACING • ANIMAZIONE IN TEMPO REALE
- ROTAZIONI E TRASFORMAZIONI NELLO SPAZIO
- GRAFICA IN C E IN BASIC

CONTIENE DISCO 3½"



GRUPPO EDITORIALE
JACKSON

AMIGA grafica 3D e animazione

Axel Plenge



GRUPPO EDITORIALE JACKSON
Via Rosellini, 12
20124 Milano

Titolo originale:
3D–GRAFIK UND ANIMATION

TRADUZIONE: Studio Dr. M. Padovani
REVISIONE TECNICA: Sergio Ruocco
MPAGINAZIONE ELETTRONICA: D.E.Ca – Lugo (RA)
REDATTORE DI COLLANA: Mauro Risani
COPERTINA: Emiliano Bernasconi

Indice

Capitolo 7
Un'altra applicazione: Solidi di rotazione 301

Capitolo 8
Appendice 319

PREMESSA DI UN LETTORE ESPERTO

Normalmente scrivo premesse solo per i miei libri, infatti cosa me ne viene in tasca, se lodo i prodotti della concorrenza? Cerchiamo comunque di rimanere obiettivi, ammettendo che questo libro ha guadagnato molte lodi.

Contrariamente a molti libri di grafica, in questo non vengono presentati solo un paio di programmi con i quali il lettore può giocare, ma viene trasmessa conoscenza.

Spesso per paura di spaventare l'eventuale compratore, molti libri nascondono il fatto che la grafica non è altro che matematica applicata. L'autore del presente libro non combatte questo dato di fatto non modificabile, bensì si addentra nel problema.

E' riuscito a spiegare, in maniera piacevolmente leggibile e comprensibile a tutti, complicate dimostrazioni matematiche. I puristi della matematica potrebbero avere obiezioni da fare in qualche punto, ma le esposizioni sono sempre obiettive e giungono con precisione allo scopo, cioè alla soluzione dal punto di vista del programma.

Anche i programmi sono sempre al massimo livello. Essi sono accuratamente sviluppati, esaurientemente commentati e forniscono soluzioni interessanti con effetti grafici fantastici.

Naturalmente i programmi arricchiscono il libro; ma io li vedo solo come un regalo utile. L'aspetto positivo più importante da valutare è che il presente testo fornisce al lettore tutte le conoscenze, a partire dalla base fino ai trucchi dei professionisti, mettendolo in grado di scrivere programmi grafici complessi.

Peter Wollschlaeger

CAPITOLO 1

Introduzione

Avete ancora qualche dubbio di avere fatto la scelta giusta al momento dell'acquisto dell'Amiga? Bene, osservate quindi le figure in questo libro.

Se invece siete convinti di aver portato a casa il miglior computer del mondo, dovete continuare sulla stessa linea e cercare un libro adeguato a tale ipermacchina. No, non guardate nello scaffale, lo avete già in mano!

E' con queste parole che potrebbe cominciare l'introduzione ad un libro di seconda scelta. Noi naturalmente vogliamo occuparci solo di prodotti di prima scelta, per cui cerchiamo di rimanere seri!

E' comunque sempre divertente riuscire a tirar fuori da una macchina di questo genere tutto ciò che è possibile: il massimo di velocità, risoluzione e colore. Questo libro vi aiuterà. Vi renderà professionisti, specialisti rispetto a tutto ciò che riguarda la grafica computerizzata.

La premessa principale è la comprensione. Il settore della matematica che assume un ruolo decisivo nella grafica viene spesso descritto come difficile e complicato. Questo libro vuole superare questi pregiudizi. Dovrete capire fin dall'inizio ciò che state leggendo. Questo è il motivo per cui ci si è occupati con tanta cura di una suddivisione sistematica degli argomenti, si sono introdotte delle spiegazioni esaurienti, anche se così facendo si è giunti spesso, inevitabilmente, ad una spiegazione doppia (meglio due che nessuna).

Introduciamo quindi il secondo principio del presente libro: elevato contenuto informativo. Quì troveremo argomenti che finora erano noti solo agli "iniziati" e agli specialisti.

I molti programmi di esempio, molto ben documentati ed istruttivi, che vi faranno contemporaneamente divertire, vi spiegheranno la teoria praticata e scoprirete che non è poi così difficile come pensavate in precedenza. Per quanto riguarda la grafica non abbiamo inventato niente di nuovo. I programmi sono scritti o in AmigaBasic o nel noto linguaggio di programmazione C, che si sta guadagnando un numero sempre maggiore di estimatori. Tutti i programmi sono pieni di commenti, e vengono descritti esaurientemente nel testo. Per gli amici del linguaggio C è sicuramente una buona notizia sapere che tutti i programmi in C possono venire compilati senza modifica sia con il compilatore Aztec che con il compilatore Lattice.

Siete diventati curiosi? Leggete fino in fondo quello che abbiamo da offrirvi. Prima di tutto ci occuperemo di alcune caratteristiche grafiche di base dell'Amiga, le cui conoscenze sono la premessa inevitabile per i rimanenti capitoli del libro. Infatti cercheremo di non dare nulla per scontato. Troverete quindi informazioni sui colori, sulle risoluzioni grafiche, sui modi grafici, ecc. Contemporaneamente impararete come poter utilizzare nella maniera più efficace tali caratteristiche dal BASIC e dal C per Amiga.

Dopo questa breve introduzione, addentriamoci nella questione. Si tratta di rappresentare, dapprima in maniera bidimensionale, le figure che in seguito ci accompagneranno nei capitoli relativi alla tridimensione: punti, linee, cerchi, ellissi e "quel che c'è dietro". Il punto chiave è costituito dalle cosidette trasformazioni: imparerete come poter ingrandire , spostare, ruotare o specchiare degli oggetti su un piano. Le basi matematiche per fare ciò vengono spiegate esaurientemente ed in modo comprensibile, e vi garantisco che alla fine del capitolo avrete veramente imparato qualcosa.

Il cosidetto Clipping di linee, per alcuni concetto sconosciuto, per altri un fantasma spaventoso, così importante per le finestre, vi apparterrà completamente, così come oggi vi appartiene la semplice deformazione delle zone video. Quest'ultimo è noto anche con il nome di "Monna Lisa in proiezione su una bottiglia di Coca Cola", cioè come avvolgere un'immagine grafica attorno ad un cilindro o ad una sfera.

Dopo l'introduzione dei concetti base, particolarmente importanti per la grafica tridimensionale, ci occuperemo della grafica tridimensionale vera e propria. Ci addentreremo nei dettagli dei sistemi di coordinate tridimensionali, nonché di come organizzare in maniera ottimale un mondo tridimensionale, come memorizzarlo e trattarlo.

Che cosa si intende con la parola proiezione centrale, o con il concetto di proiezione parallela? Come programmare dei grafici tridimensionali con ingrandimenti, rotazioni ecc. nello spazio in BASIC, e come in C? Come simulare un osservatore, come farlo entrare nello scenario di un mondo tridimensionale e molto altro. Si tratta di programmi che dovremo veramente imparare.

Nel Quinto Capitolo ci si occupa invece di rendere invisibili linee e superfici nascoste: vengono presentate alcune soluzioni del problema delle "Hidden Lines and Surfaces" sotto diversi punti di vista. Troverete anche un plotter di funzioni a tre dimensioni, assolutamente completo, che vi entusiasmerà, nonché naturalmente ogni conoscenza di base necessaria.

Sarete entusiasti anche della rappresentazione realistica di oggetti tridimensionali con superfici nascoste mostrata dai programmi, includendo una sorgente di luce posizionabile liberamente.

Se a questo punto pensate che non è possibile fare nient'altro di meglio, sono costretto purtroppo a deludervi. Tenetevi stretti!
Avete mai programmato da soli una grafica tridimensionale di elevato realismo tenendo conto di diverse fonti di luce, formazioni di ombre, riflessioni luminose a specchio e diffuse, oggetti completamente riflessi da più superfici o corpi trasparenti? Conoscete la differenza tra oggetti lucidi e opachi, lucidi, metallici e non metallici? Conoscete le immagini da voi create? La tecnica del "Ray-tracing" lo rende possibile: create il vostro mondo e riformulate le leggi della natura. Le immagini che finora avete ammirato in diapositive, potranno in futuro venire programmate da voi stessi. Sono sicuro che questa sezione del libro non ha affascinato solo me. Infine potrete produrre anche un "Juggler", ancora più realistico.

Se a quel punto sarete ancora vivi, non dovrete trascurare il Settimo Capitolo, che si occupa di rappresentazioni realistiche dei cosidetti solidi di rotazione. Le fonti di luce multiple e le ombreggiature saranno anche per voi degli strumenti di lavoro familiari.

Cosa sarebbe un libro senza appendice? Nell'appendice troverete, riunite tutte le nozioni di base della matematica comprese le funzioni trigonometriche. Il requisito minimo per la comprensione é la conoscenza dell'algebra elementare.

Nell'appendice troverete inoltre una lista completa di tutte le funzioni di Library utilizzate nel testo, con i parametri necessari per il loro uso e la descrizione.

L'opera è completata da una esauriente bibliografia.

CAPITOLO 2

Elementi di base della grafica per Amiga

E' superfluo sottolineare il fatto che l'Amiga possieda delle caratteristiche grafiche eccezionali. Dobbiamo invece esaminare attentamente come farne uso in qualità di programmatori.

Nel presente capitolo vengono illustrati tutti gli elementi necessari per produrre della grafica con l'Amiga. Si comincia dalle possibilità grafiche di base (risoluzione, colori, ecc.) passando ad una breve descrizione di come poter gestire, in AmigaBasic o in C, le diverse operazioni grafiche. Ciò ha a che vedere con i comandi BASIC per la grafica e naturalmente con le Library grafiche. Troverete inoltre gli elementi di base per i numerosi programmi esempio del libro. Naturalmente potete saltare il presente capitolo, se siete già in possesso delle conoscenze necessarie.

2.1 I colori dell'Amiga

Al fine di comprendere la gestione dei colori nell'Amiga, e in genere in tutti gli elaboratori e monitor RGB, dobbiamo intraprendere una breve digressione nella scienza generica dei colori.

Sicuramente tutti voi conoscete la tecnica di produrre con gli acquerelli, con i colori di base rosso, blu e giallo, qualunque colore desiderato. Il viola per esempio viene prodotto miscelando il rosso e il blu, il verde si ottiene dal blu e dal giallo ed il nero dalla mescolanza di tutti e tre i colori. Se un viola non è sufficientemente scuro, aggiungeremo ancora un po' di blu, eccetera. Quindi, quanto più colore base aggiungiamo, tanto più scuro diventerà il colore risultante. Tale fenomeno viene chiamato "miscelazione sottrattiva".

Un altro tipo di miscelazione dei colori è rappresentato dalla cosidetta "miscelazione additiva". Il fenomeno si manifesta nella miscelazione di luci colorate. Anche in questo caso esistono tre colori di base, ma qui si tratta di rosso, verde e blu, dalla cui combinazione è possibile ottenere tutte le altre tinte (tutto ciò non è un fenomeno naturale inspiegabile, ma ha solo a che vedere con il modo nel quale i nostri occhi identificano i colori). Quindi, nel caso della miscelazione additiva, quanto più intensamente vengono mischiati i colori di base, tanto più chiaro sarà il colore risultante (ecco perché viene chiamata miscelazione additiva). Una combinazione di parti uguali di tutti e tre i colori di base, per esempio, forma il bianco. Lo schizzo che segue mostra quali colori si ottengono dalla miscelazione dei colori base.

I moderni monitor e televisori a colori utilizzano questo principio. Ogni singolo punto sullo schermo è composto in realtà da tre punti colorati posti l'uno vicino all'altro (provate ad osservare attentamente), di colore rosso, verde e blu.

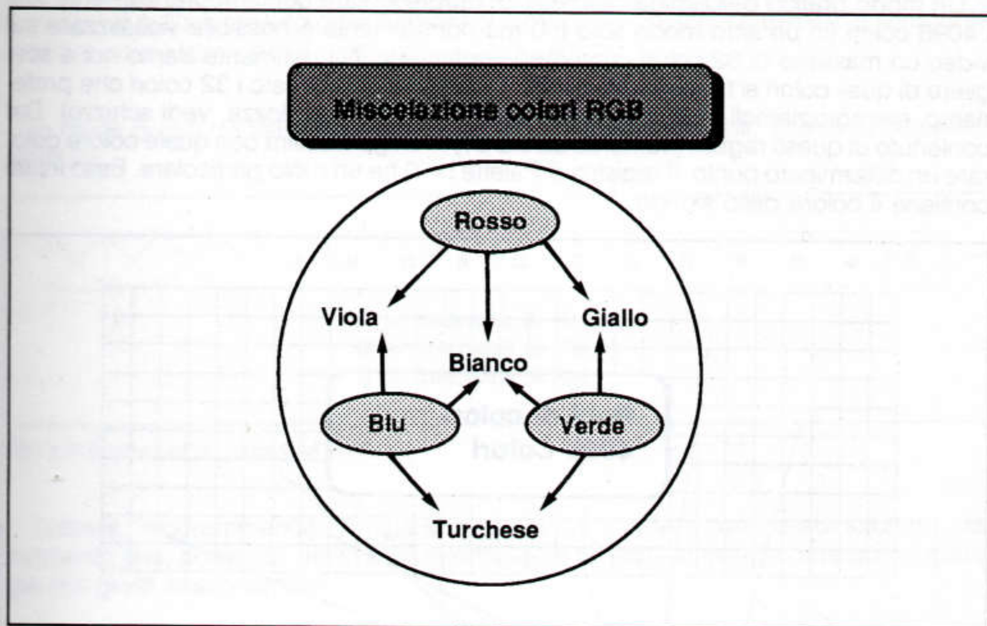


Fig. 2.1 Miscelazione colori additiva

A causa della distanza del nostro occhio da questo insieme di punti, noi registriamo un solo punto, il cui colore non è altro che la somma dei tre colori dei singoli punti. Se per esempio un punto deve apparire bianco, tutti e tre i punti singoli avranno la stessa intensità. Per ottenere il giallo lavorano solo il rosso ed il verde, mentre per il nero tutte e tre le "luci" sono spente.

Dal momento che il nostro Amiga deve produrre delle immagini su monitor a colori, esso dovrà anche sapere con quale intensità dei colori base comporre il colore di un punto. Il programmatore ha la possibilità di specificare esattamente queste informazioni. L'Amiga per ogni colore base dispone di una gamma di sedici livelli di intensità, a partire dal nero fino alla massima intensità luminosa. Ora siamo in grado di scegliere un qualunque colore risultante da una combinazione di questi tre colori base. Sarà possibile effettuare una scelta fra una gamma di 4096 combinazioni (16 intensità per ogni colore di base danno come risultato $16 \times 16 \times 16 = 4096$ colori). Ne deriva una versatilità nel colore molto utile per le immagini computerizzate e che, quasi incredibile, verrà completamente sfruttata. Speciali terminali grafici molto costosi permettono spesso una scelta di colori ancora più ampia (più di un milione di colori), cosa che si ripercuote sulla qualità dell'immagine.

Un modo grafico dell'Amiga permette di rappresentare contemporaneamente tutti i 4096 colori (in un altro modo solo 64) ma normalmente è possibile visualizzare sul video un massimo di 32 colori contemporaneamente. Naturalmente siamo noi a scegliere di quali colori si tratterà, per tali 32. Dopo aver selezionato i 32 colori che preferiamo, memorizziamoli in 32 cosidetti registri di Palette (tavolozza, vedi schizzo). Dal contenuto di questi registri (numerati da 0 a 31) l'Amiga stabilirà con quale colore colorare un determinato punto. Il registro di Palette nr. 0 ha un ruolo particolare. Esso infatti contiene il colore dello sfondo.

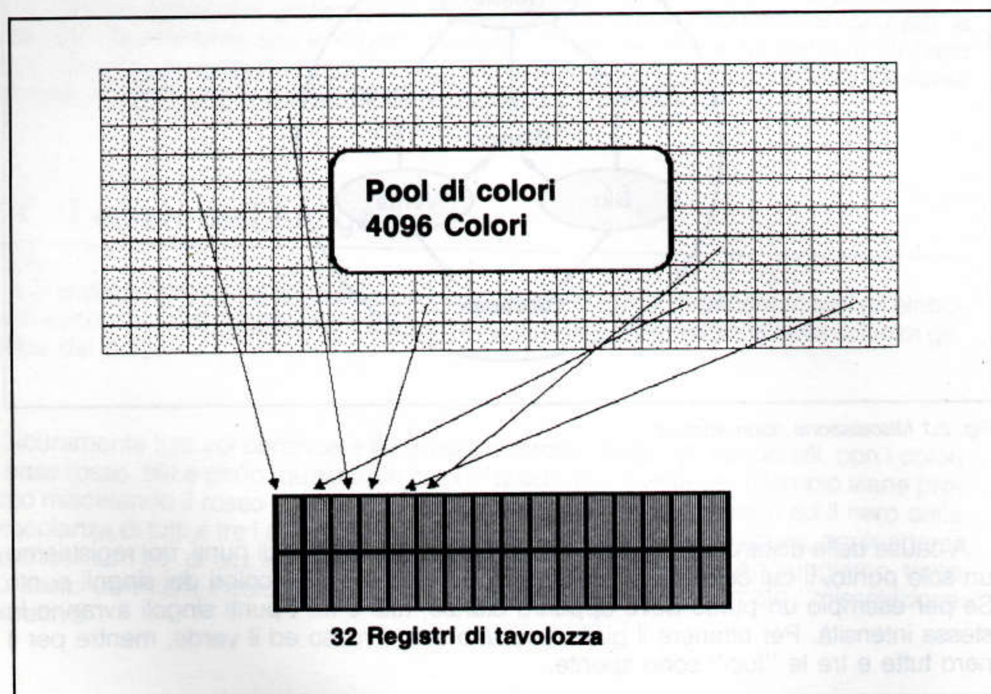


Fig. 2.2 Scelta di 32 registri di tavolozza

Ogni qualvolta tracceremo un punto, dovremo indicare il registro della Palette contenente le intensità dei colori base (RGB) per il colore di quel punto. I 32 registri di Palette hanno in un certo senso il ruolo di 32 pennelli, i quali vengono immersi in un determinato colore, quindi utilizzati per dipingere.

Vediamo quindi come memorizzare un colore in tali registri di Palette.

In primo luogo si deve indicare il valore di intensità (0-15) dei tre colori di base rosso, verde e blu. In AmigaBasic si ottiene molto semplicemente tramite il comando PALETTE. Con esso è possibile indicare i valori dell'intensità sotto forma di numeri tra 0,00 e 1,00 (quota percentuale).

In C, utilizzando la funzione della Graphics Library **SetRBG4()** oppure **SetRGB4MC()** (torneremo sull'argomento delle funzioni di Library nel presente capitolo), è sufficiente indicare solo i valori per il rosso, per il verde e per il blu.

In realtà questi tre valori vengono riuniti in un registro a 32 bit, avente la seguente struttura:

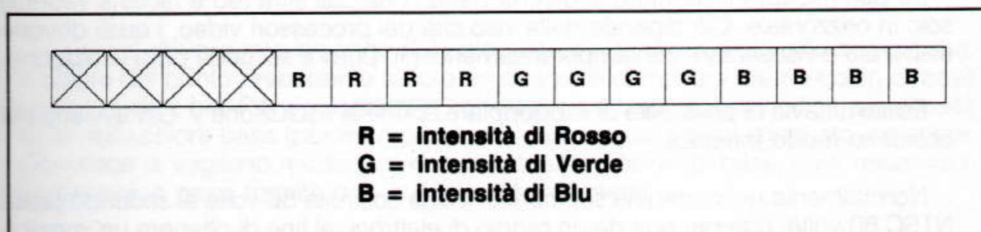


Fig. 2.3 Struttura di un registro di tavolozza

Tuttavia, programmando l'Amiga si avrà poco a che fare con questa struttura, dal momento che, come già detto, utilizzeremo delle funzioni specifiche che si occupano già dei giusti assegnamenti.

2.2 Risoluzioni grafiche e sistemi di coordinate

Anche l'Amiga, come molti altri computer, possiede diverse risoluzioni grafiche per diversi scopi. Esso utilizza un sistema molto flessibile e ricco di variazioni per la selezione della risoluzione, dei colori e del modo grafico giusto.

a) Risoluzioni standard:

Con il concetto di ''risoluzioni standard'' si intendono due modi diversi, facili da usare e di conseguenza utilizzati spesso. Sono i:

Bassa Risoluzione:	320 × 256 punti (320 × 200 sotto NTSC) max. 32 colori contemporaneamente
Risoluzione Alta:	640 × 256 punti (640 × 200 sotto NTSC) max. 16 colori contemporaneamente

(Quando si scrive 320 × 200 si intende 320 punti in direzione orizzontale, cioè direzione x, e 200 punti in direzione verticale, cioè direzione y. NTSC è la norma televisiva americana, ma noi lavoriamo con il sistema PAL). Il Workbench per esempio lavora normalmente in alta risoluzione, nella quale sono possibili 80 caratteri per riga. La bassa risoluzione viene applicata principalmente quando si vuole risparmiare spazio in memoria (ogni punto in più sul monitor costa naturalmente più memoria video oppure quando si vogliono avere più colori. Approfondiremo meglio in seguito questo aspetto. Naturalmente le capacità del monitor (televisore) rivestono un ruolo molto importante nella scelta del modo giusto.

A questo proposito c'è da fare attenzione al fatto che in alta risoluzione ogni punto è largo la metà ma alto uguale a quelli del modo 320 x 200. La bassa risoluzione, al contrario, rappresenta dei punti quasi quadrati. Per questo motivo possono manifestarsi delle distorsioni.

b) Modo Interlace:

Le due risoluzioni standard si differenziano l'una dall'altra (ed è facile a vedersi) solo in orizzontale. Ciò dipende dalla velo cità dei processori video, i quali devono elaborare e visualizzare contemporaneamente più punti a seconda della risoluzione.

Esiste tuttavia la possibilità di raddoppiare anche la risoluzione y. Ciò avviene nel cosidetto modo Interlace.

Normalmente un'immagine sul monitor viene costruita 50 volte al secondo (sotto NTSC 60 volte) riga per riga da un raggio di elettroni, al fine di ottenere un'immagine il più possibile priva di sfarfallio. Nel modo Interlace, al contrario, accade quanto segue: un'immagine visualizzata verrà dapprima scandita una riga sì e una riga no. Quindi fra due righe resta sempre una riga vuota. Alla costruzione successiva dell'immagine, vengono completate le righe intermedie mancanti. Di conseguenza un'immagine completa ha bisogno di due processi prima che tutti i punti vengano rappresentati (e ciò accade 25 volte (sotto NTSC 30 volte) al secondo). Per questo motivo si perviene, a seconda della scelta dei colori, ad uno sfarfallamento più o meno forte dell'immagine, che non è piacevole a vedersi (consiglio: diminuire il contrasto al minimo, oppure utilizzare colori con poco contrasto, anche un paio di occhiali da sole può contribuire).

E' possibile ampliare ambedue le risoluzioni standard:

bassa risoluzione, interlace:	320x512 (oppure 320x400) max. 32 colori contemporaneamente
risoluzione alta, interlace:	640x512 (oppure 640x400) max. 16 colori contemporaneamente

Le applicazioni sono chiare: una elevata risoluzione produce una migliore vista di insieme e molti dettagli. In particolare per programmi CAD, per calcoli di tabelle oppure per elaborazione di testi si tratta di vantaggi decisivi. Un ulteriore vantaggio del modo Interlace: il numero dei colori disponibili non diminuisce rispetto ai modi standard. In caso di foto del monitor, lo sfarfallamento non ha nessuna controindicazione, in quanto di solito in tali casi si lavora con tempi di esposizione di circa 1 secondo.

c) Modo Hold-and-Modify (HAM):

Si tratta di un modo specifico dell'Amiga molto interessante. Infatti con esso si ha la possibilità di rappresentare contemporaneamente sullo schermo tutti i 4096 colori possibili. Per ottenere ciò, tuttavia, è necessario osservare alcune limitazioni:

il modo HAM funziona solo sotto due risoluzioni:

Hold-and-Modify:	320x256 (NTSC: 320x200) 320x512 (NTSC: 320x400)

Per quanto concerne la colorazione dei singoli punti, si deve distinguere tra due casi, che possono presentarsi l'uno accanto all'altro nella stessa immagine. Da un lato abbiamo la possibilità di scegliere normalmente il colore per un punto da 16 registri di Palette (in maniera simile al modo alta risoluzione). Dall'altro invece è possibile utilizzare dei colori che non si trovano in questi 16 registri, nella maniera seguente:

Il colore di un punto dipende dal colore di quello che si trova alla sua sinistra. Il colore del punto precedente è noto, in quanto è composto dai tre colori di base rosso, verde e blu. Per il colore del punto attuale sarà possibile modificare l'intensità di un solo colore base (per esempio rosso), ma ciò può venire effettuato a piacere. Se invece si vogliono modificare due o tutti e tre i colori di base, sarà necessario procedere a passi tramite uno o due punti intermedi.

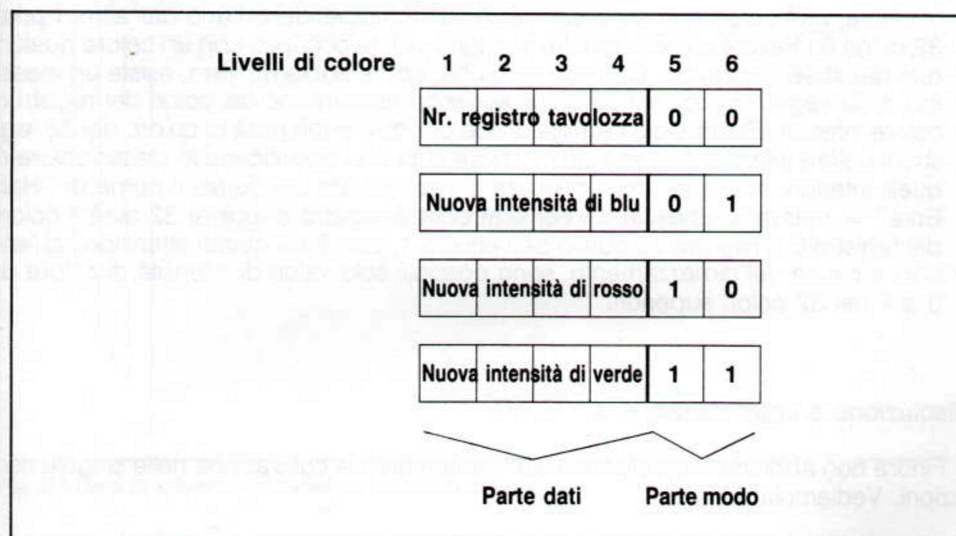


Fig. 2.4 Principio del modo Hold-and-Modify

E' necessario fare attenzione al fatto che il primo punto di una riga appare sempre nel colore dello sfondo (contenuto del registro di Palette 0). Quindi non viene mantenuto il colore dell'ultimo punto della riga precedente.

Quindi la gestione di questo modo non è poi così semplice. Ma proprio per immagini che devono contenere molti colori diversi oppure ombreggiature, è l'unica possibilità che abbiamo.

d) Modo Extra-Half _ Brite:

I progettisti del nuovo Amiga hanno introdotto un ulteriore modo, particolarmente

adeguato alla produzione di sofisticati effetti grafici: il modo Extra-Half⊇Brite. Tale modo non funziona con i primi modelli delle serie 1000 e 500. Esso è utilizzabile in due diversi livelli di risoluzione:

Standard Extra-Half⊇Brite: 320x256 (320x200)
(64 colori contemporaneamente
Interlace Extra-Half⊇Brite: 320x512 (320x400)
(64 colori contemporanemente)

Si tratta quindi di una variazione della bassa risoluzione. In questo nuovo modo grafico sono possibili contemporaneamente sullo schermo 64 colori diversi in totale, invece dei 32 colori massimi a 320x256. E' chiaro che ci deve essere anche un lato negativo: infatti tale modo necessita di molta memoria, che potrebbe dover essere ampliata.

Inoltre, i 64 colori non sono completamente indipendenti l'uno dall'altro. I primi 32 colori (la Palette colori normale) possono venire occupati con un colore qualunque dei 4096 disponibili. Dal momento che, come abbiamo visto, esiste un massimo di 32 registri di colori, i 32 colori superiori risulteranno dai colori dei registri di colore inferiori. Per fare ciò l'Amiga divide per due le intensità di colore dei 32 registri di colore inferiori. I 32 registri di colore superiori specificano lo stesso colore di quelli inferiori, solo con una luminosità dimezzata (da cui deriva il nome di "Half-Brite" = mezza luminosità). Di conseguenza il registro di colore 32 avrà il colore del registro 0, il registro 33 quello del registro 1, ecc. Fare quindi attenzione al fatto che, a causa del dimezzamento, sono possibili solo valori di intensità di colore da 0 a 7 nei 32 colori superiori.

Risoluzione e organizzazione del colore:

Finora non abbiamo approfondito sufficientemente la colorazione nelle singole risoluzioni. Vediamola meglio.

In tutti i modi (ad esclusione del Ham e Half⊇Brite) è possibile determinare quanti colori si vogliono rappresentare contemporaneamente sullo schermo. In ciò, si hanno a disposizione i seguenti valori:

max. 2 colori
max. 4 colori
max. 8 colori
max. 16 colori
max. 32 colori (solo per bassa risoluzione)

Maggiore sarà il numero di colori che si vogliono rappresentare contemporaneamente, tanto maggiore sarà lo spazio di memoria necessario per una singola immagine. Anche la velocità di rappresentazione sullo schermo diminuirà all'aumentare del numero di colori.

I colori si riferiscono come noto ad un registro di Palette, cioè un punto con colore 5 riceve il colore che si trova in quel momento nel registro di Palette 5. Nella memoria video questo numero deve venire naturalmente indicato sotto forma binaria (5 corrisponderebbe quindi alla combinazione di bit: %101). Quanti più bit sono necessari per l'indicazione del colore di un singolo punto (in caso di 5: 3 bit), tanta più memoria sarà necessaria in totale. La memoria video è organizzata nei cosidetti Bit-Plane (o piani di immagine).

Ogni Bit-Plane riceve 1 bit per ogni punto del monitor, il quale, insieme con i bit degli altri Bit-Plane, indica la colorazione del punto. Il Bit-Plane 0, quindi, fornisce tutti i bit di 0 del numero di registro del colore, il Bit-Plane 1 tutti i primi bit ecc. (vedi figura 2.5).

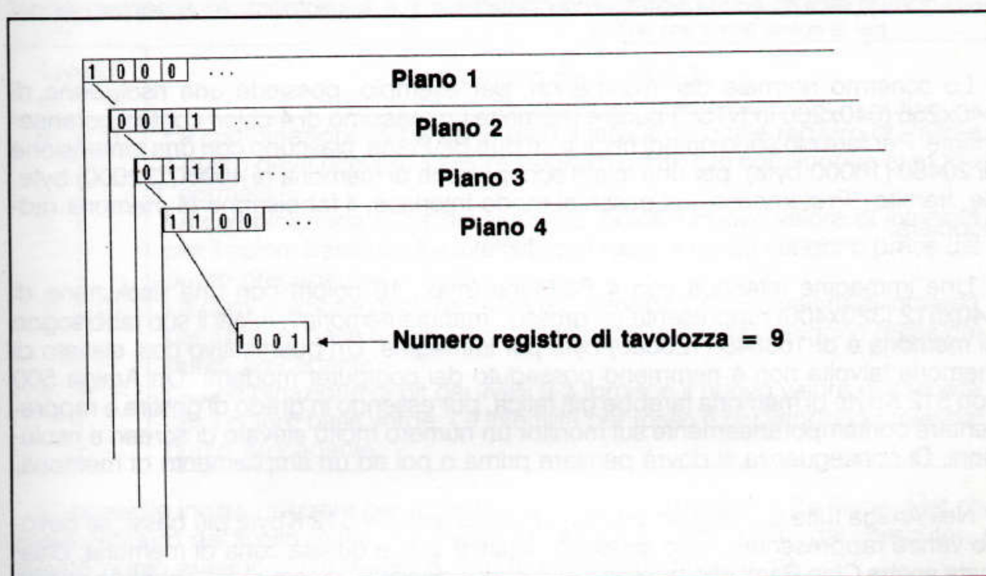


Fig. 2.5 Piani di schermo (bitplanes) e decodifica colori

Esempio: ipotizziamo che il punto in alto a sinistra del monitor debba ricevere il colore dal registro di Palette 5. Quindi nel Bit-Plane 0 avremo un 1 nel primo bit, nel Bit-Plane 1 uno 0, nel Bit-Plane 2 di nuovo un 1.

A seconda di quanti colori vogliamo rappresentare contemporaneamente, avremo bisogno di un diverso numero di Bit-Plane:

max. 2 colori - 1 Bit-Plane
max. 4 colori - 2 Bit-Plane
max. 8 colori - 3 Bit-Plane
max. 16 colori - 4 Bit-Plane
max. 32 colori - 5 Bit-Plane

Ogni ulteriore Bit-Plane necessita di spazio di memoria aggiuntivo. Tale spazio aggiuntivo dipende naturalmente dal numero dei punti rappresentabili sullo schermo, cioè

dalla risoluzione. La seguente tabella fornisce le esigenze di memoria di un Bit-Plane nelle diverse risoluzioni:

Risoluzione	Spazio di memoria per Bit-Plane in byte
320x256 (320x200)	10240 (8000)
320x512 (320x400)	20480 (16000)
640x256 (640x200)	20480 (16000)
640x512 (640x400)	40960 (32000)

Tabella 2.1: Esigenza di memoria dei livelli di colore a seconda della risoluzione (i valori tra parentesi valgono per la norma americana NTSC)

Lo schermo normale del Workbench, per esempio, possiede una risoluzione di 640x256 (640x200 in NTSC) punti e permette un massimo di 4 colori contemporaneamente. Per fare ciò sono quindi necessari due Bit-Plane, ciascuno con una dimensione di 20480 (16000 byte), per una totale occupazione di memoria di 40960 (32000) byte. Se, tramite "Preferences", si passa al modo Interlace, il fabbisogno di memoria raddoppierà.

Una immagine Interlace con 4 Bit-Plane (max. 16 colori) con una risoluzione di 640x512 (320x400) rappresenta un grosso "mangiamemoria", infatti il suo fabbisogno di memoria è di 163840 (128000) byte per immagine. Un quantitativo così elevato di memoria talvolta non è nemmeno posseduto dai computer moderni. Un Amiga 500 con 512 Kbyte di memoria farebbe già fatica, pur essendo in grado di gestire e rappresentare contemporaneamente sul monitor un numero molto elevato di screen e risoluzioni. Di conseguenza si dovrà pensare prima o poi ad un ampliamento di memoria.

Nell'Amiga tutte le immagini possono trovarsi solo nei 512 Kbyte più bassi, se devono venire rappresentate sullo schermo. Infatti è solo a questa zona di memoria, chiamata anche Chip Ram, che possono accedere i diversi "Custom-chip" i quali, in qualità di coprocessori, si occupano anche della visualizzazione e del tracciamento rapido dei grafici. Di conseguenza, anche in caso di ampliamento della memoria di lavoro, il numero degli screen sarà limitato, anche se non strettamente. In realtà i programmi possono funzionare anche in questa zona, ma, a seconda delle attività dei coprocessori, potranno essere più lenti anche del 30%, dal momento che la CPU 68000 deve continuamente attendere i coprocessori mentre accedono a tale memoria. L'ideale sarebbe avere della memoria aggiuntiva, la cosidetta Fast-Ram (già di serie nell'Amiga 2000) nella quale i programmi possono girare indisturbati.

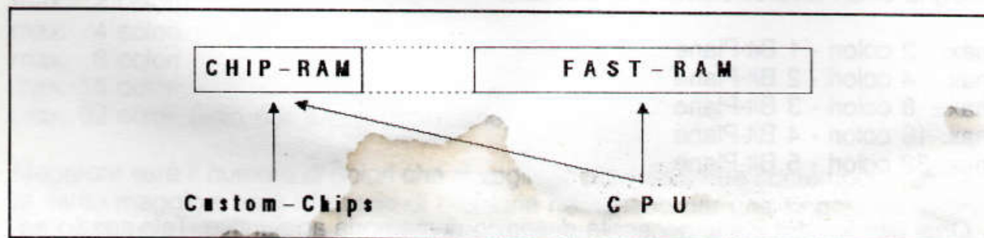


Fig. 2.6 Accesso alla memoria della CPU e dei Coprocessori

Organizzazione dei colori in modo Ham:

Come già citato, l'organizzazione dei colori riveste un ruolo del tutto particolare nel modo "Hold-and-Modify". Tale ruolo può venire descritto brevemente come segue:

Abbiamo già visto il principio su cui si basa. Il colore di un punto viene determinato sulla base del colore del punto precedente, e uno dei tre colori di base viene variato. Ma come facciamo a comunicare ciò al computer?

La risposta è data dalla organizzazione in memoria di un'immagine Ham: ogni immagine Ham possiede infatti 6 Bit-Plane (da 0 a 5). I primi 4 e gli ultimi 2 Bit-Plane fanno gruppo a sè, mentre il 4 e il 5 indicano cosa fare con i 4 bit dei numeri 0-3:

Livello 4	Livello 5	Compito dei livelli 0-3
0	0	La combinazione di bit dei livelli 0-3 indica da quale registro di Palette il punto deve ricevere il colore (registro 0-15). Ciò corrisponde al modo "normale".
0	1	La combinazione di bit dei livelli 0-3 indica il nuovo valore di intensità per il colore base blu. Le intensità per rosso e verde vengono prese dal punto precedente.
1	0	La combinazione di bit dei livelli 0-3 fornisce il nuovo valore di intensità per il colore base rosso. Le intensità per blu e verde vengono prese dal punto precedente.
1	1	La combinazione di bit dei livelli 0-3 fornisce il nuovo valore di intensità per il colore base verde. Le intensità per rosso e blu vengono prese dal punto precedente.

E' possibile inoltre utilizzare per un'immagine Ham anche solo 5 Bit-Plane. I bit che proverrebbero dal livello mancante (livello 5) vengono messi a "0" dal computer. A questo punto avremo a disposizione solo le combinazioni 00 e 10.

Come già detto in precedenza, il primo punto di una riga appare sempre nel colore dello sfondo. Ciò dovrà essere tenuto in considerazione per eventuali modifiche di intensità sulla singola riga video.

Organizzazione dei colori in modo Half_Brite:

Il modo Half_Brite, già descritto, necessita anch'esso com l'Ham di una gestione particolare. La sua struttura è comunque un po' più semplice. In aggiunta ai 5 livelli colore massimi possibili per la bassa risoluzione, se ne aggiunge un sesto. Ciò è anche logico dal momento che vogliamo codificare 64 diversi colori. Questi sei livelli di colore sono attivati automaticamente nel modo Half_Brite, quindi non sarà possibile disattivarne qualcuno.

Con i singoli bit dei sei livelli, il computer procederà esattamente come nel caso di risoluzione normale. E' con i primi 5 di essi che esso determinerà il numero del registro contenente il colore del punto in questione. Nel caso che il bit del VI livello sia ad 1, l'intensità del colore del punto sarà dimezzata.

2.3 Verso la grafica in Basic

L'AmigaBasic possiede già un numero molto elevato di comandi per la produzione o manipolazione della grafica. In tal modo è possibile rappresentare sullo schermo senza problema dei semplici grafici. Nell'appendice troveremo un breve riassunto dei comandi grafici più importanti dell'AmigaBasic. Ulteriori informazioni sono reperibili sul manuale dell'AmigaBasic che dovrebbe accompagnare ogni Amiga.

Ciò che a noi interessa in maniera particolare è la chiamata delle funzioni interne all'Amiga da BASIC. Il sistema operativo del nostro elaboratore ci mette a disposizione numerose funzioni grafiche nella sua Graphics Library (biblioteca di sottoprogrammi grafici). Talvolta i comandi in BASIC accedono a tale library (per esempio al fine di tracciare una riga o di riempire superfici).

Tuttavia alcune funzioni della Library Graphics non sono utilizzabili da BASIC. In ogni caso la chiamata diretta delle funzioni di sistema operativo presenta un vantaggio in velocità.

A ciò si aggiunge il fatto che l'AmigaBasic ipotizza di funzionare con la versione 1.1 del sistema operativo. Nella versione 1.2, implementata anche nel vostro computer, sono state aggiunte alcune funzioni, come il tracciamento di cerchi o di ellissi. L'AmigaBasic si serve in tali casi di routine proprie molto più lente.

Cosa sono le Library?

L'Amiga mette a disposizione del programmatore funzioni per gli scopi più svariati, con le quali egli potrà gestire il proprio computer, produrre dei grafici, oppure creare delle finestre sul video. Queste funzioni sono contenute in diverse biblioteche. Ecco un elenco delle Library più importanti per i nostri fini:

Biblioteche matematiche:

— mathffp.library (ROM-resident)
— mathtrans.library (Disk-resident)
— mathieeedoubbas.library (Disk-resident)

Tali library contegono numerose funzioni di calcolo algebrico e trigonometrico, ecc. Solo "mathffp.library" si trova già nella ROM. Le altre biblioteche si trovano sul disco di Workbench e devono venire caricate dal sistema operativo in caso di necessità, comunque in maniera totalmente trasparente per l'utente.

Biblioteca grafica:

— graphics.library (ROM-resident)

La biblioteca grafica contiene funzioni elementari per tracciamento di linee, ellissi, superfici e i comandi generali di animazione (Sprites, Bobs, ecc.)

Biblioteca Intuition:

— Intuition.library (ROM-resident)

Essa viene utilizzata al fine di gestire gli screen, le window, i menu, i requester (finestra di richiesta) ecc. anche per i propri programmi.

Biblioteca di sistema (Multitasking):

— exec.library (ROM-resident)

Questa biblioteca gestisce tutte le funzioni di sistema essenziali per quanto concerne il multitasking, la gestione della memoria, delle library, delle code, ecc.

Oltre a ciò esistono anche delle biblioteche come "exec_support.library", "clist.library", "layers.library", "dos.library", "icon.library" e "diskfont.library". Il sistema di biblioteche è "aperto", cioè biblioteche a piacere possono venire aggiunte in qualunque momento. Ciò naturalmente offre al programmatore possibilità enormi.

Prima di poter chiamare le singole funzioni di una biblioteca, questa deve venire aperta. Di conseguenza, una biblioteca che si trovi eventualmente su disco, dovrà prima venire caricata in memoria, quindi dovrà venire assegnato spazio in memoria per l'esecuzione del comando in questione.

Nel BASIC, ciò accade tramite il comando LIBRARY, per esempio:

```
LIBRARY "graphics.library"
```

In questo caso viene aperta la biblioteca grafica. Le singole funzioni devono venire chiamate come un normale programma in assembler con CALL:

```
CALL SetDrMd(adr,mod)
```

La Label di salto (in questo caso "SetDrMd") punta esattamente all'indirizzo della routine da utilizzare alla quale verranno trasmessi i parametri che si trovano tra parentesi.

Con un LIBRARY CLOSE vengono richiuse tutte le library aperte in BASIC (max. 5).

Il comando LIBRARY, però, fa anche qualcos'altro. Affinché all'AmigaBasic siano noti i nomi delle diverse funzioni di biblioteca (esempio "SetDrMd") esso carica un cosidetto file.bmap da disco, il quale contiene informazioni su quali funzioni sono disponibili, come si chiamano, quali parametri devono venire trasferiti e in quale sequenza. Per la "graphics.library" il file bmap ad esso appartenente si chiama "graphics.bmap". Questo file deve essere naturalmente disponibile sul dischetto corrente, diversamente otterremo una segnalazione di errore.

Di conseguenza per ogni biblioteca che vogliamo chiamare dal BASIC deve esistere su dischetto un file bmap di questo genere. Osserviamo quindi il dischetto dell'AmigaBasic (EXTRAS). Nella directory "BasicDemos" si trovano già i file "graphics.bmap" e "exec.bmap". Copiamo quindi sul nostro dischetto di lavoro tali file. Ma come fare per ottenere gli altri file bmap?

Nel nostro EXTRAS troveremo un'altra directory chiamata "FD1.2". Dal CLI sarà possibile far listare il contenuto di tale directory. In essa troveremo dei file con nomi molto particolari, quali " mathieeedoubbas_lib.fd" oppure "mathieeesingbas_lib.fd". A tutti loro è comune il suffisso ".fd". In tali casi si tratta di file in ASCII puro. Essi contengono, come i file bmap, tutte le informazioni sui nomi di funzione, sulla chiamata e sul trasferimento di parametri, ma in questo caso sono editabili in ASCII.

Tramite il programma in BASIC "ConvertFD" (presente anch'esso nel "BasicDemos", ved. Intestazione di programma) sarà possibile trasformare tali file FD in file bmap. Per ogni biblioteca avremo bisogno solo del programma "ConvertFD", quindi dovremo fornire i nomi esatti, dopodiché otterremo i file bmap necessari per tutte le biblioteche (naturalmente non avremo bisogno di farlo per le biblioteche "graphics.bmap" e "exec.bmap").

Si consiglia di effettuare tale conversione una volta per tutte e di memorizzare il risultato sul dischetto di lavoro, al fine di avere sempre pronte tutte le biblioteche.

Le circa 500 funzioni delle diverse library, e forse qualcuna in più, sono d'altra parte descritte nei 4 libri relativi all'Amiga: "ROM-Kernel Reference Manual: Libraries and Devices", "ROM-Kernel Reference Manual: Exec", "Intuition Reference Manual" e "Hardware Reference Manual" della Addison Wesley. Nel primo libro (Libraries and Devices) è presente inoltre un elenco completo di tutte le funzioni di biblioteca, con descrizione, sintassi di chiamata e passaggio dei parametri. Si tratta quindi di uno strumento indispensabile per il programmatore esperto. In ogni caso, quando useremo delle funzioni nel libro, le troveremo complete di spiegazioni esaurienti.

Vediamo ora un piccolo programma di esempio, che produce sullo schermo una serie di ellissi. Esso appare qui in due versioni. La prima fa uso del comando originale BASIC **CIRCLE**, la seconda della funzione **DrawEllipse()**. E' opportuno paragonare la velocità di disegno:

Programma 1:

```
FOR x=1 TO 400 STEP 2
  xc% = x
  yc% = 40*SIN(x/30)+100
  CIRCLE (xc%,yc%),50,,,,.6
NEXT x
```

Programma 2:

```
LIBRARY "graphics.library"

FOR x=1 TO 400 STEP 2
  xc% = x
  yc% = 40*SIN(x/30)+100
  CALL DrawEllipse(WINDOW(8),xc%,yc%,50,35)
NEXT x

LIBRARY CLOSE
```

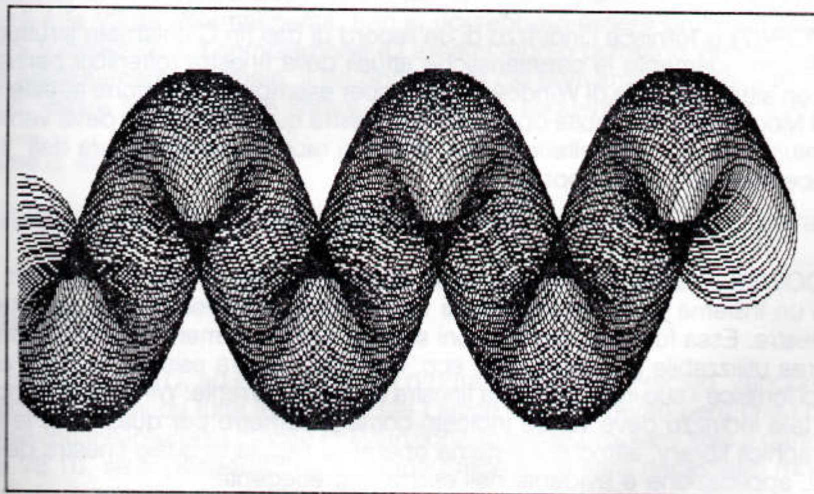


Fig. 2.7 Il DrawEllipse dell'AmigaBasic

A questo proposito dobbiamo addentrarci anche nella programmazione degli screen (schermi) e delle window (finestre). Questo compito è particolarmente semplice in AmigaBasic: il comando SCREEN inizializza un nuovo schermo con una risoluzione ed un numero di colori a piacere e gli attribuisce un numero di riferimento. Con SCREEN CLOSE è possibile chiudere l ; schermo (Ved. manuale del BASIC per Amiga). Purtroppo

in AmigaBasic è attualmente possibile produrre solo schermi con un massimo di 200 righe (400 in modo Interlace).

I comandi grafici in BASIC si riferiscono spesso ad una finestra. Sarà pertanto necessario aprire anche una nuova finestra all'interno di un nuovo screen. Ciò è realizzabile utilizzando il comando WINDOW. WINDOW apre una finestra con dimensioni a piacere. A tale finestra è possibile aggiungere la DRAG-BAR (sbarra di trascinamento) ed i diversi Gadget (chiusura, ingrandimento ecc.). Ogni finestra si riferisce al numero di identificazione dello SCREEN, nel quale essa deve venire aperta. Anche la finestra riceve un numero di identificazione, che sarà poi necessario per identificarla prima di inviarle i comandi di Output. Troveremo sufficienti esempi per la programmazione di screen e di window negli ultimi capitoli del libro. Anche le finestre, in AmigaBasic, hanno un massimo di 200 righe (400 in modo Interlace).

Tramite l'importante funzione BASIC **WINDOW(n)** è possibile ottenere informazioni aggiornate sulla finestra di output corren te (n determina il tipo di informazione che vogliamo richiedere). A questo proposito consultare il manuale. In particolare, in questo momento ci interessano i valori 7 e 8 attribuiti a n:

WINDOW(7) ci fornisce l'indirizzo di un record di dati (in C chiamato struttura), che contiene continuamente le caratteristiche attuali della finestra (ottenibili parzialmente anche con altre chiamate di Window()), come per esempio la posizione attuale del cursore del Mouse ecc. Per molte operazioni di finestra questo indirizzo deve venire fornito alla Intuition library. Per ulteriori informazioni si raccomanda la lettura dell'"Intuition Reference Manual" (ved. sopra).

WINDOW(8) ci fornisce l'indirizzo di un ulteriore record di dati: la RastPort. La RastPort è un insieme molto importante di variabili che rappresentano le caratteristiche delle finestre. Essa fornisce informazioni sulla posizione in memoria attuale della finestra, l'area utilizzabile per disegnare, ecc. Per ogni finestra esiste una RastPort. Window(8) ci fornisce il suo indirizzo per la finestra di output corrente. Window(8) è importante perché tale indirizzo deve venire indicato come parametro per quasi tutte le funzioni della graphics library, affinché il sistema operativo sappia in quale finestra deve disegnare. L'applicazione è evidente nell'esempio precedente.

2.4 Verso la grafica in C

Prima di leggere questo capitolo sarà oppurtuno occuparsi brevemente di quanto descritto nella gestione della grafica sotto BASIC, dal momento che molte parti di esso sono utilizzabili anche per il C.

Sotto C la gestione della grafica, degli screen e delle window, a causa delle molte dichiarazioni, si presenta un po' più complicata di quanto non lo sia sotto BASIC. In linea di massima, però, molti aspetti sono simili. Naturalmente non sarà più necessario produrre da soli le library standard. Esse si trovano sul dischetto del compilatore C e vengono collegate automaticamente dal linker. A causa del numero molto elevato di strutture necessarie per la chiamata delle routine di library (come parametri di funzioni) le quali talvolta riflettono lo stato attuale del sistema, esistono numerosi file include per ogni biblioteca (riconoscibili sul dischetto del compilatore C dal suffisso ".h"). Essi contengono l'inizializzazione per le strutture più diverse e definiscono diverse istruzioni del preprocessore. Si consiglia di stampare tutti i file di include e di classificarli ordinatamente. In tal modo sarà possibile ricercare in qualunque momento nomi e contenuti di strutture. E' tuttavia possibile trovare i file di include commentati anche nei quattro libri precedentemente citati della Addison Wesley.

Tutti i programmi in C del libro girano, come già citato nell'introduzione, sia sul compilatore in C Aztec che con il compilatore Lattice a partire dalla versione 3.10. Al fine di esserne certi, è necessario prestare attenzione ancora ad un aspetto:

Compilatore Aztec:

La versione di cui l'autore disponeva e per la quale è possibile garantire che tutto funzionerà è la V3.4a. Normalmente, non ci dovrebbero essere problemi anche con le versioni successive. La compilazione e l'assemblaggio potranno venire effettuati in maniera assolutamente normale. Il file oggetto da ciò derivante (estensione ".o") dovrà quindi venire linkato con le library. Utilizzare la seguente sintassi di link:

```
ln filename.o -lc -lm
```

Con tale sintassi vengono linkate insieme le normali funzioni di library e la library FFP.

Compilatore Lattice:

Tutti i programmi in C, ad esclusione del programma di Ray-Tracing, vengono compilati senza problemi da tutte le versioni del Lattice. Dal momento però che i collegamenti delle routine in virgola mobile FFP funzionano senza problemi solo a partire dalla funzione V3.10, se si possiede una versione precedente sarà necessario modificare leggermente il programma di "Ray-Tracing".

A partire dalla versione V3.10, sarà possibile compilare con le seguenti sequenze di comandi:

```
lc1 -f -i:include/ -i:include/lattice filename.c
lc2 filename
blink lib:c.o+filename.o LIBRARY lib:lc.lib+
lib:amiga.lib+lib:lcmffp.lib TO filename
```

Sarà inoltre possibile dover linkare il file oggetto **Lib:Istartup.obj** invece del file **lib:c.o.**

Il seguente programma DEMO ci permetterà di addentrarci nella programmazione di Screen, Window e Grafica:

```
/**************************************/
/**                                 **/
/**    Programmazione di schermi    **/
/**     Finestre e grafica in C     **/
/**                                 **/
/**           Axel Plenge           **/
/**                                 **/
/**                                 **/
/**                                 **/
/**************************************/

#include <exec/types.h>
#include <intuition/intuition.h>

/* Dichiarazione funzioni (solo per compilatore Aztec-C: */
/* a scelta anche: #include <functions.h>             */
/*****************************************************/
VOID                     CloseLibrary();
VOID                     CloseScreen();
VOID                     CloseWindow();
VOID                     Draw();
VOID                     Exit();
struct Message *         GetMsg();
VOID                     Move();
struct Library *         OpenLibrary();
struct Screen *          OpenScreen();
struct Window *          OpenWindow();
VOID                     RectFill();
VOID                     ReplyMsg();
VOID                     SetAPen();
VOID                     SetDrMd();
LONG                     Text();
LONG                     Wait();
/*****************************************************/
```

```
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

 /* Inizializzazione struttura per nuovo Font: */
struct TextAttr NeuerFont =
{
   "topaz.font",          /* Nome del Font    */
   TOPAZ_SIXTY,           /* Dimensione font */
   FS_NORMAL,             /* Stile           */
   FPF_ROMFONT,           /* Preimpostazione */
};

 /* Struttura per inizializzazione di un
    nuovo schermo (naturalmente cio' puo'
    aver luogo anche dentro il Programma):
  */
struct NewScreen NeuerBildschirm =
{
   0,                     /* Coordinata x superiore sinistra  */
                          /* Angolo (sempre 0)                */
   0,                     /* Coordinata y superiore sinistra  */
                          /* Angolo                           */
   320,                   /* Larghezza schermo                */
   256,                   /* Altezza schermo                  */
   2,                     /* Numero immagini                  */
   0,                     /* Colore dei dettagli              */
   1,                     /* Colore delle superfici           */
   NULL,                  /* Modo grafico: 320x200            */
   CUSTOMSCREEN,          /* Tipo schermo                     */
   &NeuerFont,            /* Puntatore a propri font          */
   "Demo-Schermo",        /* Testo di intestazione schermo    */
   NULL,                  /* inutilizzato, sempre ZERO (NULL) */
   NULL,                  /* nessun BitMap proprio            */
};

/* Struttura per l'inizializzazione di una nuova finestra: */
struct NewWindow NeuesFenster =
{
   20,                    /* Coordinata x angolo super. sin.  */
   20,                    /* Coordinata y angolo super. sin.  */
   300,                   /* Larghezza finestra               */
   100,                   /* Altezza finestra                 */
   0,                     /* Colore dei dettagli              */
   1,                     /* Colore delle superfici           */
   CLOSEWINDOW |          /* Risposta per finestra chiusa     */
    NEWSIZE,              /* dimensioni finestra              */
                          /* modificate                       */
```

```
    WINDOWCLOSE |            /* Elementi e tipo finestra        */
     SMART_REFRESH |         /* da selezionare                  */
     ACTIVATE |
     WINDOWSIZING |
     WINDOWDRAG |
     WINDOWDEPTH |
     NOCAREREFRESH |
     GIMMEZEROZERO,
    NULL,                    /* Nessun gadget proprio           */
    NULL,                    /* CheckMark                       */
    "Demo-Finestra",         /* Testo di intestazione finestra  */
    0,                       /* Indirizzo struttura schermo     */
                             /* deve venire inizializzato       */
                             /* all'interno del Programma       */
                             /* dopo l'apertura di              */
                             /* uno schermo                     */
    NULL,                    /* nessuna finestra SuperBitmap    */
    100,                     /* Larghezza minima                */
    25,                      /* altezza minima                  */
    320,                     /* Larghezza massima               */
    256,                     /* Altezza massima                 */
    CUSTOMSCREEN,            /* Tipo schermo                    */
};

VOID main()
{
    struct Screen *Bildschirm;
    struct Window *Fenster;
    struct IntuiMessage *Meldung;
    struct RastPort *rasterport;

    LONG    i;
    LONG    fensterbreite,
            fensterhoehe;
    ULONG   class;

/* Apertura delle library di intuition e grafiche.      *
 * La funzione  OpenLibrary() restituisce               *
 * un puntatore alla library                            *
 * che deve venire memorizzato, affinche' il            *
 * Programma in C possa accedere ad esso. Se tale       *
 * puntatore e' uguale a zero, c'e' un errore           *
 */
```

```
IntuitionBase =
(struct IntuitionBase *)OpenLibrary("intuition.library",0L);
if (IntuitionBase == NULL)  Exit(FALSE);

GfxBase =
(struct GfxBase *)OpenLibrary("graphics.library",0L);
if (GfxBase == NULL)
{
   CloseLibrary(IntuitionBase);    /* Intuition-Library close */
   Exit(FALSE);                    /* Uscita                  */
}


/* Ora viene aperto lo schermo. La funzione            *
 * OpenScreen() fornisce un puntatore alla              *
 * Struttura di screen attuale, che dovremo             *
 * annotarci. Se e' uguale a zero, c'e' un errore       *
 */
Bildschirm =
(struct Screen *)OpenScreen(&NeuerBildschirm);
if (Bildschirm == NULL)
{
   CloseLibrary(GfxBase);          /* Graphics-Library close  */
   CloseLibrary(IntuitionBase);    /* Intuition-Library close */
   Exit(FALSE);                    /* Uscita                  */
}


/* Tramite OpenWindow viene aperta una finestra. La    *
 * funzione fornisce un puntatore alla struttura di     *
 * Window attuale. Se e' uguale a zero, c'e' un errore  *
 */
NeuesFenster.Screen = Bildschirm; /* Impostazione indirizzo della
                                     struttura di schermo          */
Fenster =
(struct Window *)OpenWindow(&NeuesFenster);
if (Fenster == NULL)
{

   CloseScreen(Bildschirm);        /* Chiusura schermo        */
   CloseLibrary(GfxBase);          /* Graphics-Library close  */
   CloseLibrary(IntuitionBase);    /* Intuition-Library close */
   Exit(FALSE);
}


/* Con cio' abbiamo una nuova finestra all'interno     *
 * un nuovo schermo. Ora, per dimostrazione, verra'     *
 * emesso un piccolo grafico con testo                  *
 */
```

```
/* Indirizzo della Rasterport: */
rasterport = Fenster->RPort;

do
{
   fensterbreite = Fenster->Width - 1;
   fensterhoehe = Fenster->Height - 1;

   SetAPen(rasterport, 2L);            /* Impostaz. colore di disegno */
   SetDrMd(rasterport, (LONG)JAM1);/* Modo di disegno su JAM1       */

   for (i=0; i < fensterbreite+1; i=i+3)
   {
      /* Posizionare il cursore grafico e tracciare la linea: */
      Move(rasterport, fensterbreite-i, fensterhoehe);
      Draw(rasterport, i, 0L);
   }

   for (i=0; i < fensterhoehe+1; i=i+3)
   {
      /* Posizionare il cursore grafico e tracciare la linea: */
      Move(rasterport, fensterbreite, i);
      Draw(rasterport, 0L, fensterhoehe-i);
   }

   /* Iscrizione testo:                 */
   SetDrMd(rasterport, (LONG)JAM2); /* Modo di disegno su JAM2  */
   Move(rasterport, fensterbreite/2-45L, fensterhoehe/2+3L);
   Text(rasterport, "Demo Finestra",13L);


   /* Attesa Messaggio:                 */
   Wait(1L << Fenster->UserPort->mp_SigBit);

   /* Elaborazione segnalazione ricevuta: */
   Meldung = (struct IntuiMessage *)
   GetMsg(Fenster->UserPort);           /* Indirizzo segnalazione */
   class = Meldung->Class;              /* Tipo di segnalazione   */
   ReplyMsg(Meldung);                   /* Restituzione segnalaz. */

   if ((class & NEWSIZE) != 0)          /* La dimensione finestra */
   {                                    /* e' stata modificata    */
      SetAPen(rasterport, 0L);          /* Colore per disegno = 0 */
      SetDrMd(rasterport, (LONG)JAM1);  /* Modo di disegno: JAM1  */
      RectFill(rasterport, 0L, 0L,      /* Cancellazione finestra */
               fensterbreite, fensterhoehe);
   }
```

```
} while ((class & CLOSEWINDOW) == 0);      /* finche' la finestra    */
                                           /* non e' chiusa          */

CloseWindow(Fenster);                      /* Chiusura finestra      */
CloseScreen(Bildschirm);                   /* e anche schermo        */
CloseLibrary(GfxBase);                     /* Graphics-Library       */
CloseLibrary(IntuitionBase);               /* Intuition-Library      */

Exit(TRUE);                                /* Uscita                 */
```

Questo programma apre una finestra in un nuovo schermo 320x200, nel quale deve venire tracciato un grafico ed un testo. La finestra è trascinabile a piacere e può venire ampliata o ristretta, mentre il suo contenuto si adeguerà in continuazione alla nuova dimensione. Al fine di terminare il programma, sarà sufficiente azionare il gadget CLOSE (quindi chiudendo la finestra).

Dal momento che vogliamo lavorare con le library Exec, Graphics e Intuition, abbiamo bisogno, per la definizione delle diverse strutture, dei file Include "exec/types.h" e "intuition/intuition.h". Quest'ultimo carica automaticamente i file Include necessari per la library Graphics.

Ora inizia la definizione delle variabili e delle strutture globali. A questo punto dobbiamo imparare a conoscere i puntatori "*IntuitionBase" e "*GfxBase", i quali forniscono gli indirizzi di strutture che vengono utilizzate per ogni library. Essi verranno in seguito utilizzati, tra l'altro, anche per richiudere le library.

Quindi inizializziamo la struttura per una nuova stringa di caratteri. La stringa di caratteri verrà ricaricata automaticamente in caso di necessità. Dal momento che vogliamo aprire un nuovo schermo, dovremo inoltre preparare una struttura per tale schermo.

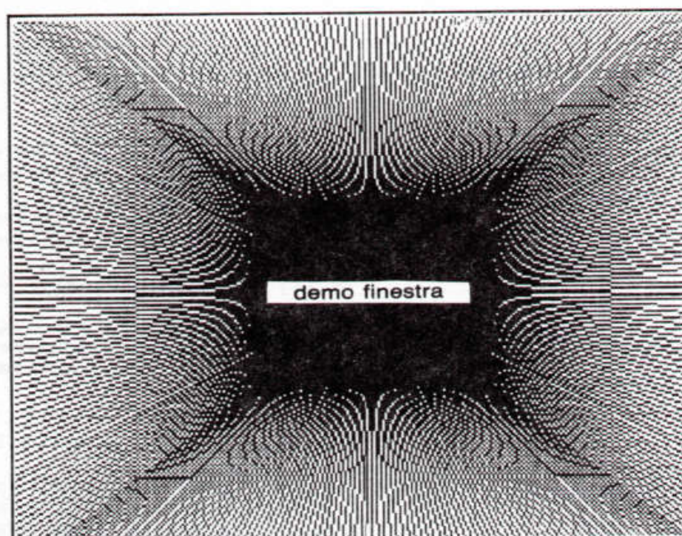


Fig. 2.8 Screen, Window e Grafica in C

Qui forniamo le diverse caratteristiche (risoluzione, dimensione, numero di colori, font, ecc.) che dovrà assumere. Naturalmente avremmo potuto inizializzare tale struttura anche nel programma, attribuendo ad ogni singolo campo il valore corrispondente. Nel nostro caso tuttavia è molto più semplice. Inoltre in questo modo non abbiamo bisogno di conoscere i numerosi nomi dei campi (i quali, come noto, si trovano nella definizione di struttura, quindi nei file Include).

Procederemo quindi nello stesso modo con la struttura NewWindow. In tal caso dobbiamo introdurre solo in seguito almeno un parametro. Si tratta dell'indirizzo della struttura di Screen corrispondente, che conosceremo solo dopo l'apertura vera e propria dello schermo. In tale struttura dobbiamo predisporre inoltre numerosi flag. Scegliamo in questo caso due flag di segnalazione (IDCMP flag). Il nostro programma dovrà ricevere la comunicazione di quando l'utente chiude la finestra (CLOSEWINDOW) e di quando la ingrandisce oppure la rimpicciolise (NEWSIZE). Il tipo di finestra verrà comunicato con un altro flag. La nostra finestra possiede inoltre un gadget di chiusura (WINDOWCLOSE), un gadget per la modifica delle dimensioni (WINDOWSIZING), per la modifica della disposizione in profondità delle finestre (WINDOWDEPTH) e per il suo posizionamento libero (WINDOWDRAG). Inoltre attiviamo SMART_REFRESH (il contenuto della finestra dovrà venire rinnovato solo in caso di ingrandimento/diminuzione), ACTIVATE (la finestra è attiva dopo l'apertura), NOCAREREFRESH (nessuna segnalazione di Refresh). Allo stesso tempo la nostra finestra è anche una cosiddetta finestra "Gimmezerozero", cioè all'interno della finestra viene aperta un'altra finestra invisibile. Quest'ultima coincide con l'intero spazio di finestra, esclusi i campi occupati dai gadget e dalla DRAG-BAR ecc. Gli output grafici vengono effettuati relativamente a tali dimensioni di finestra, evitando di disegnare sui gadget e sul contorno della finestra stessa.

A questo punto è veramente possibile cominciare. Dopo l'inizializzazione delle strutture e delle variabili locali necessarie, cerchiamo di aprire le due library Intuition-library e Graphics-library con la funzione Exec **OpenLibrary()** (la library Exec è già aperta all'accensione dell'Amiga, di conseguenza non necessita di venire aperta da noi). Annotiamoci per usi futuri i puntatori ottenuti. Se però il puntatore restituito è uguale a 0, c'è qualcosa che non ha funzionato (es. memoria insufficiente, biblioteca non disponibile, magari non presente su disco ecc.). In questo caso terminiamo semplicemente il nostro programma con la funzione **exit()**, senza tuttavia dimenticare di richiudere eventuali library già aperte.

Procederemo quindi con l'apertura dello schermo con **OpenScreen()** e della finestra con **OpenWindow()**. Ad ambedue queste funzioni forniremo semplicemente un puntatore alle strutture precedentemente inizializzate (new screen oppure new window). Riceveremo il puntatore alla struttura di finestra o di schermo di sistema. Facciamo tuttavia attenzione al fatto di aver impostato l'indirizzo della struttura dello schermo prima dell'apertura della finestra.

Con ciò dovremmo essere riusciti ad aprire una finestra in un nuovo schermo. Al fine di utilizzarla, tracciamo in essa un breve disegno. Molte funzioni grafiche necessitano dell'indicazione di un puntatore, che troveremo nella struttura di finestra, di sistema, la Rastport. La Rastport è una struttura che, semplificando, indica al sistema operativo dove deve disegnare (in quale finestra). Nel nostro programma copiamo tale puntatore nella variabile "Rasterport".

A questo punto è possibile utilizzare diverse funzioni grafiche. Alla fine del nostro disegno dovremo attendere una azione dell'utente. Ciò è ottenibile con la funzione di Exec **Wait()**, che una segnalazione nella Message-Port della finestra (durante l'attesa possono così venire eseguiti altri processi e task). Il funzionamento esatto di questa funzione esula dal contenuto del presente testo. Il tipo di segnalazione è presente anche nella porta utente della finestra. Copiamolo nella variabile "Class" e restituiamo la segnalazione con **ReplyMsg()**.

A questo punto dovremo agire a seconda del tipo di segnalazione. Se la finestra è stata ampliata o rimpicciolita (NEWSIZE) dall'utente, cancelliamo il contenuto della finestra e ridisegnamo tutto da capo. Se è stato clickato il gadget di chiusura (CLOSEWINDOW), ciò per noi significherà: Fine del programma. Chiudiamo quindi la finestra, lo schermo e le library (fare attenzione alla sequenza) e terminiamo con **Exit()**.

Come abbiamo già visto, è possibile inserire numerosi flag nelle strutture di nuovi schermi e finestre. Nella pagina seguente troviamo tutti i flag possibili per queste due strutture. Ogni nome di flag rappresenta una combinazione di bit. Nel caso in cui si vogliano impostare due o più flag contemporaneamente, questi dovranno venire congiunti con una operazione OR (in C "|") (vedi programma esempio). Per quanto concerne le singole strutture, e in particolare le strutture vere e proprie di schermo e di finestra (struct Screen e struct Window), le quali vengono inizializzate tramite la chiamata di OpenScreen() oppure OpenWindow(), sarà possibile rilevarne gli estremi dai diversi file di Include dal dischetto del compilatore C. Passiamo quindi alle tabelle:

a) Flag per la struttura di schermo:

ViewModes (modi grafici):

NULL	— Grafica a bassa risoluzione a 320 punti per riga
HIRES	— Grafica ad alta risoluzione a 640 punti per riga
LACE	— Attivazione del modo Interlace
HAM	— Attivazione del modo Hold-and-Mody
EXTRA_HALF_BRITE	— Attivazione del modo Extra_Half_Brite
DUALPF	— Modo Dual-Playfield: modo speciale, nel quale uno schermo appare sotto un altro schermo
PFBA	— Flag per Dual-Playfield: esso determina quale dei due schermi deve apparire sotto l'altro.
SPRITES	— Si utilizzeranno degli Sprite nello schermo
VP_HIDE	— Lo schermo viene prodotto, ma non rappresentato sul monitor
GENLOCK_VIDEO	— Modo speciale per una interfaccia Genlock. Una immagine video viene sovraimpressa sostituendo i pixel del colore di sfondo.

ScreenTypes (Tipi di schermo):

WBENCHSCREEN	— Screen di Workbench
CUSTOMSCREEN	— Nuovo schermo
SHOWTITLE	— Visualizzazione della title bar (riga di titolo)
BEEPING	— Quando lo schermo lampeggia, il flag viene impostato da Intuition
CUSTOMBITMAP	— Viene utilizzato un BitMap predefinito.

b) Flag per la struttura di finestra:

IDCMP-Flag (flag di comunicazione di Intuition):

I flag IDCMP, posti nella struttura di finestra, specificando se il programma deve venire informato nel caso in cui si presenti un determinato evento. Sarà possibile attendere tale evento con la funzione **Wait()** (vedi programma esempio). Per venire informati invece di quale evento si tratta, usiamo il campo Class della struttura IntuiMessage (vedi programma esempio). Anche qui viene di nuovo impostato il flag IDCMP dell'evento corrispondente. Le singole informazioni relative all'evento sono presenti, a seconda del tipo di segnalazione, negli altri campi di struttura come Code, MouseX, MouseY ecc. Vediamo quindi i flag:

MOUSEBUTTON	— Segnalazione, nel caso in cui venga azionato un pulsante del Mouse. L'indicazione di quale tasto del Mouse si tratti è contenuta nel campo Code della struttura IntuiMessage (SELECTDOWN, SELECTUP, MENUDOWN, MENUUP).

MOUSEMOVE	— Segnalazione nel caso in cui il Mouse venga mosso (da impostare solo con REPORTMOUSE).
DELTAMOVE	— come MOUSEMOVE. Tuttavia vengono comunicate anche coordinate relative del Mouse.
GADGETDOWN	— Segnalazione di quando viene selezionato un gadget
GADGETUP	— Segnalazione di quando un gadget viene rilasciato
CLOSEWINDOW	— Segnalazione di quando viene chiusa una finestra
MENUPICK	— Segnalazione di quando viene selezionato una opzione del menu. Il codice di menu si trova nel campo Code della struttura IntuiMessage
MENUVERIFY	— Segnalazione di quando l'utente sta per selezionare una opzione del menu (segnalazione prima della selezione)
REQSET	— Segnalazione di quando viene aperto il primo Requester nella finestra
REQCLEAR	— Segnalazione di quando viene chiuso l'ultimo Requester nella finestra
REQVERIFY	— Segnalazione prima della chiusura dell'ultimo Requester nella finestra
NEWSIZE	— Segnalazione di quando viene modificata la dimensione di finestra
REFRESHWINDOW	— Segnalazione di quando devono venire rinnovate alcune parti della finestra
SIZEVERIFY	— Segnalazione di quando l'utente vorrebbe ingrandire una finestra (La segnalazione precede all'ingrandimento)
ACTIVEWINDOW	— Segnalazione di quando una finestra viene attivata
INACTIVEWINDOW	— Segnalazione di quando una finestra viene disattivata
RAWKEY	— Segnalazione di pressione di un tasto
NEWPREFS	— Segnalazione di quando vengono modificate le Preferences
DISKINSERTED	— Segnalazione di quando viene inserito un dischetto
DISKREMOVED	— Segnalazione di quando viene tolto un dischetto
INTUITICKS	— Segnalazione del Timer di Intuition (circa 10 volte al secondo)

Flag di finestra:

WINDOWSIZING	— La finestra deve possedere un gadget di ingrandimento/riduzione
WINDOWDRAG	— La finestra deve poter venire spostata
WINDOWDEPTH	— La finestra deve possedere dei gadget di variazione di profondità
WINDOWCLOSE	— La finestra deve possedere un gadget di chiusura
SIZEBRIGHT	— Il gadget delle dimensioni deve trovarsi al bordo destro
SIZEBBOTTOM	— Il gadget delle dimensioni deve trovarsi al bordo inferiore

SMART_REFRESH	— Dopo la scoperta da parte di un'altra finestra, questa finestra deve venire ridisegnata automaticamente
SIMPLE_REFRESH	— Dopo la scoperta da parte di un'altra finestra, il programma deve effettuare da solo la ricostruzione della parte di finestra scoperta
SUPER_BITMAP	— La finestra è solo una parte di un'area maggiore (Super BitMap) e verrà "refreshed" automaticamente
BACKDROP	— Finestra di sfondo (si trova sempre sotto tutte le altre finestre)
REPORTMOUSE	— Ogni movimento del Mouse deve venire segnalato (ved. anche IDCMP flag)
GIMMEZEROZERO	— All'interno di una finestra viene aperta una seconda finestra invisibile. Tutte le coordinate di disegno si riferiscono alla finestra interna. Vantaggio: le coordinate per esempio 0,0 non si troveranno più nella zona bordo della finestra (DRAG-BAR)
BORDERLESS	— La finestra viene rappresentata senza bordo
ACTIVATE	— La finestra viene attivata non appena viene aperta
WINDOWACTIVE	— Questo flag viene impostato da Intuition quando la finestra è quella attiva
INREQUEST	— Questo flag viene impostato da Intuition quando la finestra si trova in modo Request
MENUSTATE	— Questo flag viene impostato da Intuition quando è attiva la finestra con il menu inserito
NOCAREREFRESH	— Nessuna segnalazione di quando viene effettuato il Refresh.

CAPITOLO 3

Cominciamo gradualmente:

Operazioni bidimensionali

Prima di scontrarci con il mondo spaziale, con tutte le sue altezze e profondità, dovremmo osservare brevemente le tecniche grafiche essenziali per le due dimensioni. Molte di esse possono venire trasferite semplicemente nel mondo tridimensionale.

Certamente molti di tali elementi vi sono già noti. Cerchiamo quindi di partire dagli aspetti noti e di ricercare ciò che è ignoto. Contemporaneamente ci procureremo gli elementi matematici di base, ai quali in seguito non potremo più rinunciare. Grafica è matematica, ed è un dato di fatto che probabilmente non vi farà piacere.

Cerchiamo tuttavia di non spaventarci, esistono metodi e sistemi per descrivere in maniera comprensibile (e qui sta la differenza rispetto alla scuola) anche la noiosa matematica scolastica.

Cerchiamo di approfondirne gli argomenti pian piano e cominciamo con:

3.1 Elementi grafici di base: Punto, Linea, Cerchio, Ellisse

Cerchiamo di prendere confidenza con gli elementi di base di ogni grafico. Occupiamoci prima di tutto del punto. Dal momento che lo schermo del nostro computer, cosa non così ovvia, è composto da singoli punti, il punto rappresenta obbligatoriamente la più piccola unità grafica. Ogni linea, ogni cerchio, ogni altra figura non è composta nient'altro che da molti piccoli punti colorati.

Ciò tuttavia non significa che i punti siano le unità grafiche più maneggevoli. E' molto più efficace e più comodo comporre i grafici con linee. Vedremo infatti che se ne fa un uso maggiore, essendo molto facile tracciare delle linee. Esse, sia matematicamente che graficamente, sono molto semplici da maneggiare, in breve, esse rappresentano l'utensile grafico.

Dal momento che è abbastanza dispendioso, in termini di tempo, tracciare cerchi o ellissi con una velocità accettabile, troveremo quasi tutte le istruzioni per disegnare tali figure nei comandi di un buon interprete per BASIC. D'altra parte, in pochissimi sistemi operativi troviamo un supporto per la grafica (ma anche per altri scopi) così esteso, completo e semplice da utilizzare come quello fornito dal nostro Amiga.

Il seguente programma in BASIC dovrebbe fornire alcuni begli esempi per il disegno di tali figure. Per quanto concerne gli ellissi, è necessario fare attenzione anche a quanto detto nel capitolo "Verso la Grafica in Basic".

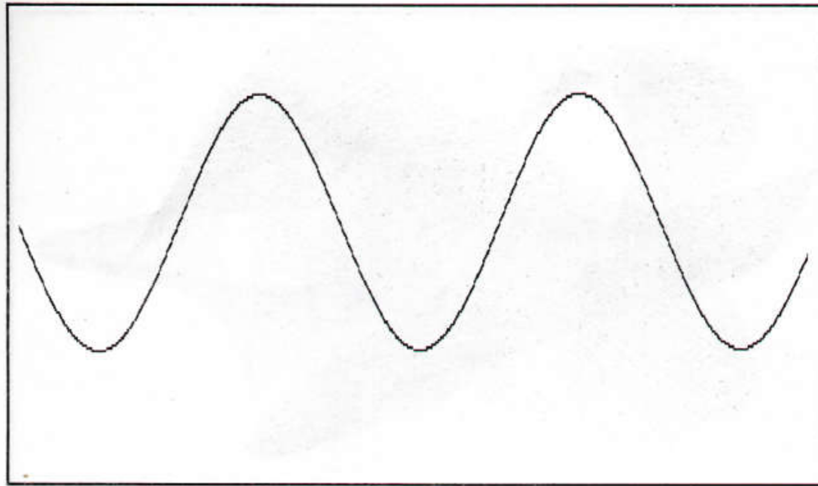


Fig. 3.1 Punti 1

```
' **********************
' **                  **
' **  Punti, Linee,   **
' **  Cerchi, Ellissi **
' **                  **
' **********************

' Punti:

COLOR 2,0
FOR x=0 TO 639
  PSET (x,50*SIN(x/40)+100)
NEXT x

WHILE INKEY$=""
  SLEEP
WEND
```

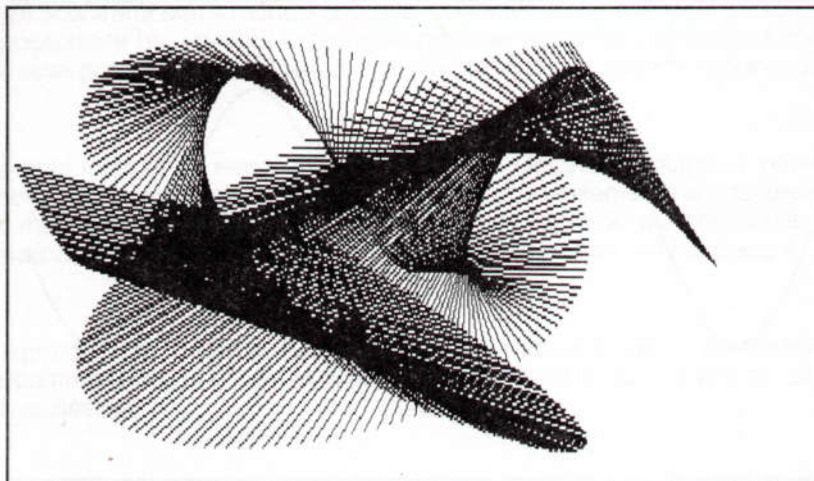


Fig. 3.2 Linee

```
' Linee:

COLOR 3,2
CLS
FOR x=0 TO 550 STEP 2
  x1 = x
  y1 = 40*SIN(x/40)+60
  x2 = 200*SIN(x/40)+250
  y2 = 80*SIN(x/50)+90
  LINE (x1,y1)-(x2,y2)
NEXT x

WHILE INKEY$=""
  SLEEP
WEND

' Cerchi ed ellissi:

COLOR 3,0
CLS
```

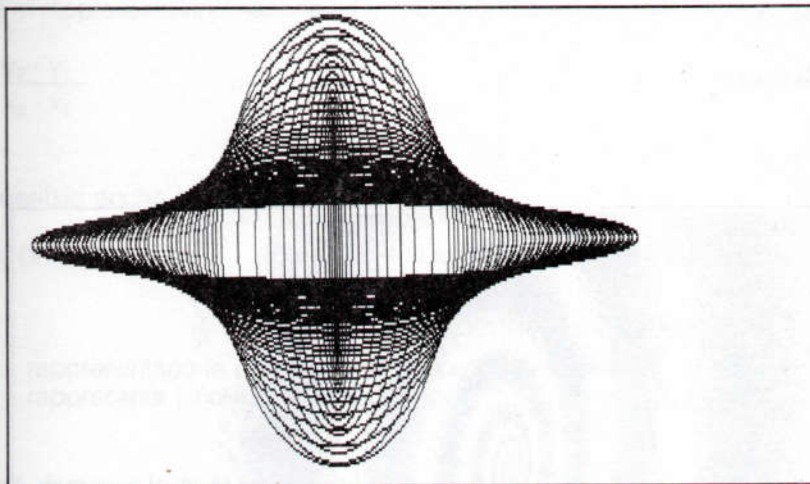


Fig. 3.3 Cerchi ed ellissi

```
FOR x=1 TO 40 STEP .7
  CIRCLE (260,90),40+5*x,,,,100/x^2
NEXT x

WHILE INKEY$=""
  SLEEP
WEND


'Ancora Punti:

COLOR 2,0
CLS
FOR x=1 TO 250
  FOR y=-100 TO 100
    w = ATN(y/x)
    r = SQR(x*x+y*y)
    h = .5 + .5*SIN(2*w+LOG(r)*10)
    IF h>=RND(1) THEN
      PSET (250-x,100+y)
      PSET (249+x,100-y)
```

```
    END IF
  NEXT y
NEXT x

WHILE INKEY$=""
  SLEEP
WEND
```

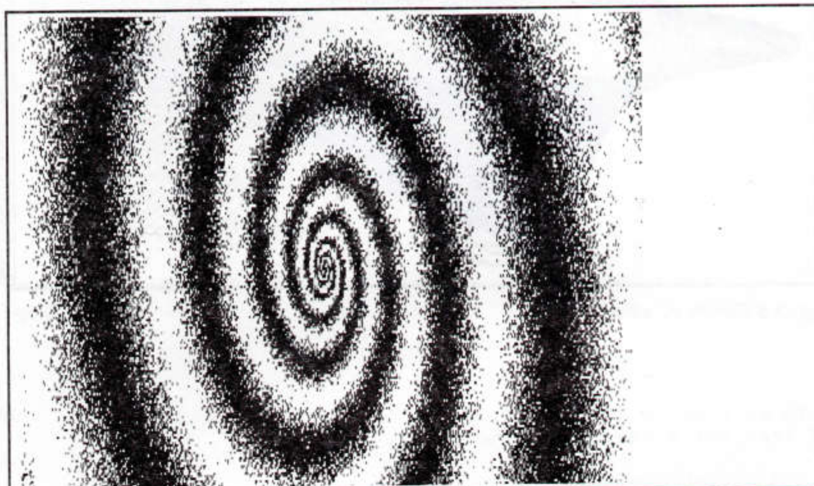


Fig. 3.4 Punti 2

Non vorremmo spiegare ulteriormente questo programma, in quanto ci porterebbe fuori argomento. Dato che si serve solo come suggerimento, osserviamolo semplicemente.

Spesso, anche nel presente libro, ci ritroveremo con il compito di dover tracciare non solo una linea (possibile anche tramite i comandi corrispondenti), ma di dover calcolare punti di intersezione con altre linee (o piani), di dover tagliare delle linee o simili. Per fare ciò abbiamo bisogno di una buona infarinatura in matematica.

Naturalmente l'equazione di una retta è essenziale, e chiunque l'avrà vista almeno una volta nella vita:

a*x + b*y + c = 0 (Forma implicita)

oppure:

y = m*x + n (Forma esplicita)

dove m rappresenta l'inclinazione ed n il punto di intersezione con l'asse y, quindi vale:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

E' possibile anche la seguente forma:

$y = m^*(x - x_1) + y_1$ (Equazione di una retta passante per un punto)

dove:

x_1, y_1 rappresentano le coordinate di un punto della retta
m rappresenta l'inclinazione (ved. sopra)

Da ciò deriva la formula seguente:

$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$ (Equazione della retta passante per due punti)

Un'ultima forma non molto diffusa è la seguente:

$\frac{x}{a} + \frac{y}{b} = 1$ (Forma di intersezione con gli assi)

Questa retta taglia l'asse x nel punto P (a,0) e l'asse y nel punto Q (0,b).

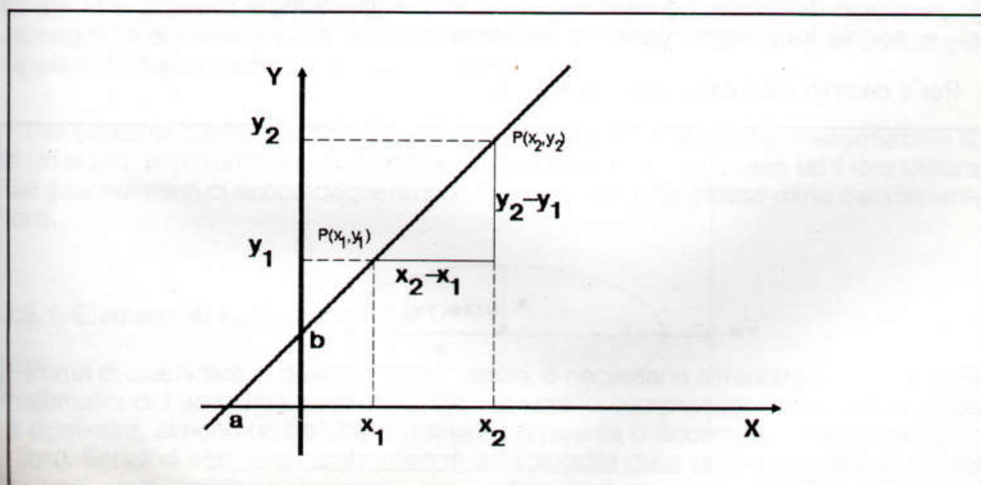


Fig. 3.5 Equazione della retta

Naturalmente esistono anche alcune altre formule, che tuttavia in questa fase non ci interessano. Incontreremo molto spesso alcune di queste equazioni, ecco perché le rappresentiamo molto sommariamente. Infatti, al momento dell'utilizzo, la singola formula verrà spiegata ulteriormente.

Anche per quanto concerne i cerchi e le ellissi (il cerchio non è altro che un caso speciale di ellisse) esistono delle equazioni matematiche che forse sono un pochino meno conociute:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad \text{(Formula implicita)}$$

dove:

a = raggio dell'ellisse in direzione x
b = raggio dell'ellisse in direzione y

mentre per il cerchio vale a = b = r, per cui:

$$\frac{x^2 + y^2}{r^2} = 1$$

Un'altra formula molto utilizzata è la seguente:

d = a * cos(w)
y = b * sin(w) (Forma parametrica)

dove:

a = raggio dell'ellisse normalizzata nell'origine in direzione x
b = raggio dell'ellisse normalizzata nell'origine in direzione y
w = angolo fra il raggio generico dell'ellisse e l'asse x

Per il cerchio vale naturalmente: a = b

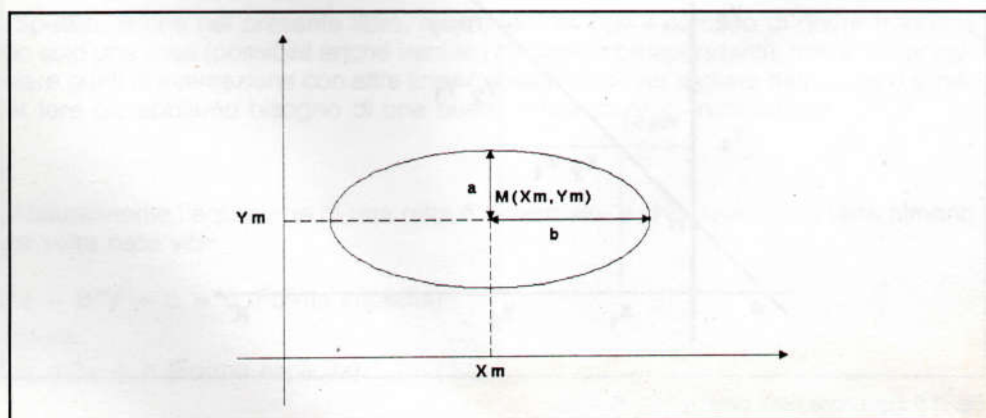


Fig. 3.6 Equazione dell'ellisse

Dopo questa breve divagazione entriamo direttamente nell'argomento, che occuperà tutto il capitolo. Nel caso in cui si desideri apprendere qualcosa in più e di più dettagliato sugli argomenti appena introdotti, si potrà fare riferimento all'appendice. In essa sono contenuti ulteriori chiarimenti.

3.2 Trasformazioni in 2 dimensioni

Nel nostro percorso verso mondi spaziali, le cosidette trasformazioni rivestono un ruolo importante. Con la parola trasformazione si intende, nel senso più ampio, una qualunque trasformazione di una immagine già esistente. Ipotizziamo di aver tracciato una piccola immagine (una casa o simili) sullo schermo. Conosciamo le coordinate dei diversi vertici ed i punti che devono venire collegati l'uno con l'altro tramite segmenti. Ora vogliamo delle operazioni matematiche semplici che ci rendano possibile poter modificare questa casa tramite un programma, senza dover modificare a mano le coordinate, una volta stabilite, spostarla sul video, ingrandirla o ridurla. Inoltre vogliamo ruotarla e specchiarla, in breve: trasformarla.

Le trasformazioni rivestono un ruolo essenziale in molte applicazioni. Gli architetti apprezzeranno molto il fatto di poter osservare le loro opere anche sotto un altro angolo di visuale, oppure quello di poter ingrandire i dettagli. Sicuramente anche i disegnatori di cartoni animati sono alla ricerca di una tecnica che renda loro possibile spostare o ruotare delle figure.

E' per questo motivo che è importante tracciare un'immagine sul video non solo punto per punto, cosa che viene resa possibile molto semplicemente con programmi di grafica come il "Deluxe Paint" o simili. E' molto più importante per noi che un'immagine sia composta da una serie di linee, i cui punti finali sono a noi noti grazie alle loro coordinate. Ora, abbiamo memorizzato le coordinate per il nostro piccolissimo "mondo delle immagini" e di conseguenza possiamo modificarle in qualunque momento tramite manipolazioni matematiche, cioè trasformazioni.

Nel presente capitolo ci occuperemo solo dell'effettuazione di tali trasformazioni su di un piano. Impareremo quindi i procedimenti matematici necessari ed il loro utilizzo. Tali procedimenti ci accompagneranno in seguito per una grossa parte del presente libro.

3.2.1 Elementi di base matematici

Prima di esaminare le diverse trasformazioni, è necessario affrontare alcuni elementi matematici di base. Sarà assolutamente necessario comprendere e essere in grado di dominare, almeno un po', tali argomenti, in quanto ci accompagneranno per tutto il libro. Senza di essi, una comprensione approfondita delle singole operazioni e delle manipolazioni grafiche è pressoché impossibile. Sarà necessario prestare particolare attenzione alla moltiplicazione delle matrici. Per noi questa è una delle operazioni più importanti.

In questa sede le cosidette matrici ci terranno occupati molto concretamente. Il calcolo delle matrici non è molto complicato, contrariamente a quanto si senta spesso dire in giro. Esso assume tuttavia un significato enorme per ogni tipo di trasformazione, in quanto rappresenta il sistema più semplice per la loro realizzazione.

Se, arrivati al termine della lettura delle pagine seguenti, non si sarà ancora in grado di comprendere l'utilizzo di quanto in esse appreso, sarà assolutamente naturale. Per poter cominciare a capire, sarà necessario attendere di essere arrivati alla fine del capitolo relativo alle due dimensioni.
Il concetto di matrice rappresenta per noi un ordinamento rettangolare di diversi numeri:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \cdots & a_{3n} \\ a_{41} & a_{42} & . & . & . & \cdots & . \\ . & . & . & . & . & \cdots & . \\ . & . & . & . & . & \cdots & . \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & a_{m5} & \cdots & a_{mn} \end{pmatrix}$$

Ogni numero a_{ik} (per esempio a_{11} oppure a_{34}) rappresenta un elemento della matrice. (Gli indici degli elementi di cui sopra servono solo per differenziare, mentre i nomi delle matrici sono normalmente delle lettere maiuscole, in questo caso A). La matrice è composta da righe (riga orizzontale di elementi) e colonne (riga verticale di elementi). Essa viene detta del tipo (m,n), dal momento che possiede m righe ed n colonne, oppure, più semplicemente matrice (m,n). Gli elementi a_{11}, a_{12}, a_{13}, ecc. (oppure brevemente: a_{1k}) rappresentano la prima riga, tutti gli elementi a_{2k} rappresentano la seconda riga, ecc. Per quanto riguarda le colonne il procedimento è analogo. Un elemento a_{ik} si trova quindi nella riga i e nella colonna k. Si tratta quindi di una matrice bidimensionale.

Il numero degli elementi in una riga può essere uguale (ma non necessariamente) al numero degli elementi di una colonna. In tale caso si tratta di una matrice quadrata a n-righe, per esempio:

$$A = \begin{pmatrix} 1 & 5 & -6 \\ -7 & 13 & 8 \\ -1 & 2 & 22 \end{pmatrix}$$

Questa matrice è quadrata e composta da 3 righe. Le matrici quadrate sono particolarmente importanti, dal momento che esse, conformemente alla definizione, sono attribuite ad un numero, il cosidetto determinante. I determinanti vengono usati per esempio quando si devono risolvere dei sistemi di equazioni a più incognite.

Un altro caso particolare di matrici sono quelle matrici che possiedono una sola riga o una sola colonna. Di conseguenza esse saranno del tipo (1,n) oppure (m,1), (vedi sopra), esempio:

A = (1 5 3 -4)

In questo caso si tratta di una matrice che possiede una sola riga (e che ha quattro "colonne"). Essa viene chiamata anche "vettore di riga". L'altro caso è il cosidetto "vettore di colonna", esempio:

$$A = \begin{pmatrix} -4 \\ 9 \\ 6 \end{pmatrix}$$

Ora, con le matrici è possibile anche effettuare dei calcoli. Infatti possiamo sommare, sottrarre e moltiplicare per un numero semplice o anche per un'altra matrice.

Senza divagare troppo, orientiamoci verso la operazione matriciale di cui abbiamo assoluto bisogno: la moltiplicazione di due matrici concatenate.

Il concetto di "matrici concatenate" significa propriamente quanto segue:

E' possibile moltiplicare l'una per l'altra due matrici, ma esse devono adempiere a determinate condizioni, cioè devono essere concatenate.

In altre parole, è possibile moltiplicare una per l'altra due matrici solo se la prima possiede un numero di colonne uguale al numero di righe dell'altra. Un esempio di una moltiplicazione possibile sarebbe quindi:

$$A * B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

A possiede tre colonne, B possiede tre righe, di conseguenza la condizione sarebbe adempiuta. Vediamo però anche che la moltiplicazione B*A non è permessa e che, anche se lo fosse, il risultato non sarebbe lo stesso di quello di A*B. La sequenza dei fattori è di conseguenza molto importante. Fare tuttavia attenzione al fatto che una moltiplicazione di matrici fra due matrici quadrate (vedi sopra) è sempre permessa. In effetti, in seguito, avremo prevalentemente a che fare con matrici quadrate.

Il risultato di una moltiplicazione di matrici è anch'esso una matrice. Quest'ultima possiede lo stesso numero di righe della prima matrice (A) e lo stesso numero di colonne della seconda matrice (B). Il risultato della moltiplicazione di cui sopra ((3,3) - Matrice) (3,2 - Matrice) sarebbe una matrice del tipo (3,2) (3 righe, 2 colonne).

Il calcolo di questa matrice di risultato C è un po' complicato. Per i lettori esperti, forniamo la formula di un singolo elemento in forma matematica abbreviata:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} * b_{jk} = a_{il} * b_{lk} + a_{i2} * b_{2k} + \ldots + a_{in} * b_{nk}$$

che significa: l'elemento c_{ik} viene determinato dalla somma di tutti gli $a_{ij} * b_{jk}$, dove $j = 1$ fino a $j = n$.

Spiegazione dei simboli:

c_{ik} un elemento della matrice di risultato C
a_{ij} un elemento della matrice A
b_{jk} un elemento della matrice B
n numero delle colonne di A e numero delle righe di B

Chiariamo meglio il tutto con il seguente esempio:

Calcoliamo, come dice la formula, un elemento c_{ik} della matrice risultato, moltiplicando tutti gli elementi delle righe i di A con gli elementi corrispondenti delle colonne k di B e sommandone i risultati:

C = A * B

$$= \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$= \begin{pmatrix} 1*1 + 2*3 + 3*5 & 1*2 + 2*4 + 3*6 \\ 4*1 + 5*3 + 6*5 & 4*2 + 5*4 + 6*6 \\ 7*1 + 8*3 + 9*5 & 7*2 + 8*4 + 9*6 \end{pmatrix}$$

$$= \begin{pmatrix} 22 & 28 & & c_{11} & c_{21} \\ 49 & 64 & = & c_{21} & c_{22} \\ 76 & 100 & & c_{31} & c_{32} \end{pmatrix}$$

Questo calcolo, per l'essere umano, può sembrare un po' complicato, ma lascia un computer completamente indifferente. Fare tuttavia attenzione al fatto che nella penultima riga l'espressione "1*1 + 2*3 + 3*5" rappresenta per esempio un solo elemen-

to. Cerchiamo ora di completare ulteriormente l'esempio. Cioè: l'elemento superiore sinistro della matrice risultato C, cioè l'elemento c_{11}, viene calcolato tramite:

$$c_{11} = a_{11}{}^*b_{11} + a_{12}{}^*b_{21} + a_{13}{}^*b_{31}$$

L'elemento c_{12} viene calcolato tramite ... ecc.

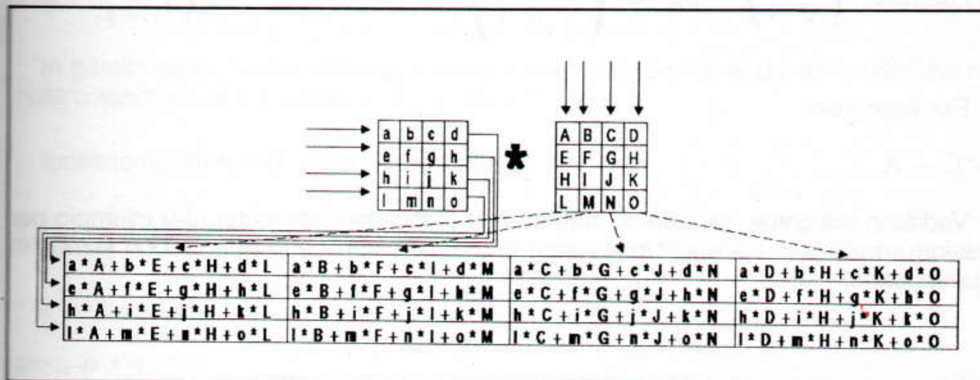


Fig. 3.7 Schema della moltiplicazione matriciale

Ora verifichiamo la nostra preparazione con i due esercizi seguenti:

Calcoliamo la matrice risultato C della moltiplicazione di matrici:

$$C = A^*B = \begin{pmatrix} 6 & 8 \\ 3 & 9 \end{pmatrix} \; ^* \; \begin{pmatrix} 1 & -5 \\ 3 & 7 \end{pmatrix}$$

$$C = A^*B = \begin{pmatrix} 3 & 5 & 2 & 8 \\ 7 & 0 & 1 & 1 \\ 5 & 5 & 3 & 2 \\ 1 & 2 & 0 & 5 \\ 4 & 4 & 4 & 4 \end{pmatrix} \; ^* \; \begin{pmatrix} 1 & 1 \\ 6 & -6 \\ 3 & 0 \\ 4 & 10 \end{pmatrix}$$

Osserviamo ancora alcune regole di calcolo. Se moltiplichiamo una riga di matrici, sarà per noi indifferente quale moltiplicazione eseguire per prima, se non ne modifichiamo la sequenza; in termini matematici:

$$A^*(B^*C) = (A^*B)^*C$$

In ogni caso sarebbe errato: $A^*(B^*C) = (A^*C)^*B$

Come già determinato precedentemente, $A^*B = B^*A$ non vale, e ciò è molto importante.

Esistono alcuni tipi di matrici, per i quali è possibile moltiplicare un'altra matrice senza modificare nulla. Si tratta delle matrici quadrate nelle quali tutti gli elementi sono uguali a zero, tranne gli elementi della cosidetta diagonale principale devono contenere il valore 1; tali matrici vengono dette anche matrici unitarie quadrate E:

$$(1) \text{ oppure } \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ oppure } \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Per esse vale:

A*E = A

Vediamo ora come utilizzare le matrici nella grafica per computer, per esempio per trasformazioni (e in tali applicazioni sarà molto più semplice di quanto non sembri in questa sede):

3.2.2 Ingrandimenti/Riduzioni bidimensionali

Esistono naturalmente altre possibilità di descrivere matematicamente le diverse trasformazioni grafiche oltre alle matrici. E' stato tuttavia determinato che nessuna di queste possibilità è così intuitiva e semplice da manipolare come le matrici.

Come abbiamo già detto, vogliamo manipolare le coordinate dei singoli punti del nostro mondo da un punto di vista matematico (=trasformare). Al fine però di non sommare punti e matrici come lo si fa con mele e pere, è necessario descrivere ogni punto in un modo adeguato ad entrare in uno schema di matrici. A tale scopo, prendiamo le due coordinate x e y di un punto e vediamole quasi come una matrice (1,2). Ipotizziamo che un punto possieda le coordinate x e y, la sua matrice sarà:

P(x,y) = (x y)

In questo caso si tratta naturalmente di una definizione assolutamente arbitraria, che non deve venire dimostrata, ma che ci sarà molto utile in futuro.

Se ora vogliamo trasformare un punto, moltiplichiamolo per una cosidetta matrice di trasformazione. Se abbiamo scelto correttamente questa matrice, la matrice di risultato dovrebbe fornire le coordinate del punto trasformato. La matrice di risultato deve di conseguenza essere di nuovo del tipo (1,2) (e quindi contenere le coordinate).

Cerchiamo quindi una matrice di trasformazione T, la quale, se moltiplicata per il punto di partenza P, determini il punto trasformato P' (cioè la sua matrice):

P' = P * T

Se ora moltiplichiamo tutti i punti della nostra immagine per T, avremo trasformato tutta l'immagine. L'aspetto finale trasformato di tale immagine dipenderà naturalmente dagli elementi della matrice T. Ipotizziamo che T sia una matrice unitaria (vedi elementi di base matematici):

$$T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

In questo caso, l'intera immagine resta esattamente com'era, dal momento che nel nostro esempio di cui sopra, P è uguale a P'.

Ipotizziamo invece di utilizzare la matrice:

$$T_1 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

Il risultato sarà:

$$\begin{aligned} P' &= P * T_1 \\ &= (x\ y) * \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \\ &= (x*2 + y*0 \quad x*0 + y*1) \\ &= (2*x\ y) \\ &= (x'\ y') \end{aligned}$$

Le ultime due righe, sotto forma di parametri, saranno:

$x' = 2*x$
$y' = y$

Vediamo quindi che, grazie alla trasformazione, ogni nuova coordinata x diventa il doppio di quanto fosse in precedenza. Ciò significherà quindi una deformazione (oppure ingrandimento) in direzione x.

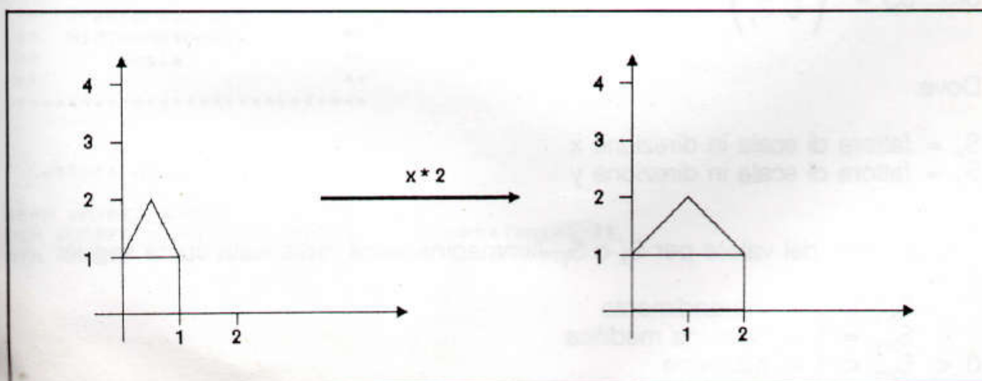


Fig. 3.8 Raddoppio di tutte le Coordinate x

Allo stesso modo, se cerchiamo una matrice che raddoppi tutte le coordinate y, e che di conseguenza deformi l'immagine in direzione y, troveremo:

$$T_2 = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

Ora, se vogliamo ingrandire (ridurre) un'immagine sia in direzione x che in direzione y, avremo bisogno della seguente formula:

$$P' = (P^*T_1)^*T_2$$

Dapprima ingrandiamola quindi in direzione x, dopodiché in direzione y. Dal momento che, secondo le regole di calcolo per le matrici, possiamo scrivere anche questa formula:

$$P' = P^*(T_1{}^*T_2)$$

sarà altrettanto semplice trovare una matrice $T_3 = T_1{}^*T_2$, la quale esegua ambedue le trasformazioni in una volta sola:

$$T_3 = T_1{}^*T_2$$

$$= \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Se generalizziamo tutto ciò, otterremo la seguente matrice di trasformazione S, per ingrandimenti/riduzioni (scala):

$$S(S_x, S_y) = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

Dove:

S_x = fattore di scala in direzione x
S_y = fattore di scala in direzione y

A seconda del valore per S_x o S_y, l'immagine verrà modificata come segue:

$S_{x/y} > 1$ — ingrandimento
$S_{x/y} = 1$ — nessuna modifica
$0 < S_{x/y} < 1$ — riduzione
$S_{x/y} < 0$ — specchiatura contemporanea sull'asse x/y

Si dovrà cercare di evitare di introdurre valori inferiori o uguali a zero per i fattori di scala, dal momento che in tal caso non si tratterà più di un semplice ingrandimento o riduzione.

Con questa semplice matrice abbiamo risolto i nostri problemi. Possiamo ora riassumere più scale, che dovranno venire eseguite una dopo l'altra, in una sola matrice a seconda delle applicazioni, oppure ottenere lo stesso risultato anche tramite diverse moltiplicazioni di matrici. Nell'elaboratore, la messa in scala di un punto P (x, y) viene eseguito con la matrice di trasformazione S secondo il seguente schema:

$$P' = P(x,y) * S(Sx, Sy)$$
$$= (x\ y) * \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$
$$= (S_x * x\ \ S_y * y)$$
$$= (x'\ y')$$

Scomponiamo di nuovo questa formula matriciale del nostro punto risultante P' in un formato di coordinate, e otterremo:

$$x' = S_x * x$$
$$y' = S_y * y$$

Queste sono le due formule che il computer dovrà calcolare. La deviazione, apparentemente complicata, per le matrici, ci sarà molto utile in futuro.

Il seguente programma in Basic dovrebbe portarci un po' più vicino all'applicazione delle trasformazioni in scala:

```
'***************************
'**  Trasformazione       **
'**  Bidimensionale       **
'**      Scala            **
'**                       **
'***************************


' Lettura di punti e linee:

READ anzp%, anzl%
DIM SHARED x.punkte(anzp%-1), y.punkte(anzp%-1)
DIM SHARED s.linien%(anzl%-1), e.linien%(anzl%-1)
```

```
'Fattori di messa in scala:
sx = 1
sy = 1

'Posizione immagine:
mx% = MOUSE(3)
my% = MOUSE(4)

'Lettura punti:
FOR i=0 TO anzp%-1
  READ x.punkte(i), y.punkte(i)
  'PRINT i": "x.punkte(i), y.punkte(i)
NEXT i

'Lettura linee (Collegamenti dei punti):
FOR i=0 TO anzl%-1
  READ s.linien%(i), e.linien%(i)
NEXT i


' Loop principale:
WHILE c<>27             'fino a ESC

  CLS                   'cancellazione schermo

  ' loop di tracciato:
  FOR i=0 TO anzl%-1
    x1% = x.punkte(s.linien%(i))
    y1% = y.punkte(s.linien%(i))
    x2% = x.punkte(e.linien%(i))
    y2% = y.punkte(e.linien%(i))
    LINE (mx%+x1%,my%-y1%)-(mx%+x2%,my%-y2%)
  NEXT i

  'ATtesa di un tasto:
  c$=""
  m%=0
  WHILE c$="" AND m%=0
    SLEEP               'Attendere e rendere possibile
                        'il Multitasking
    c$=INKEY$           'eventualmente prelevamento tasto
    m%=MOUSE(0)         'o stato del Mouse
  WEND

  sx = 1
  sy = 1
  IF m%<>0 THEN         'Azionamento tasto del Mouse?
    mx% = MOUSE(1)      'si -> nuova Posizione
    my% = MOUSE(2)
```

```
  ELSE
    c = ASC(c$)
    IF c=28 THEN         'Cursore verso l'alto?
      sy = 1.2           'si -> Ingrandimento y
    END IF
    IF c=29 THEN         'Cursore verso il basso?
      sy = .8            'si -> Riduzione y
    END IF
    IF c=30 THEN         'Cursore a sinistra?
      sx = 1.2           'si -> Ingrandimento x
    END IF
    IF c=31 THEN         'Cursore a destra?
      sx = .8            'si -> Riduzione x
    END IF
  END IF

  'Trasformazione dell'intera immagine:
  CALL transformiere.bild(sx,sy)

WEND

END

' Trasformazione di tutte le coordinate dell'immagine:
SUB transformiere.bild(sx,sy) STATIC
  SHARED i, anzp%
  FOR i=0 TO anzp%-1
    CALL s.transform(sx,sy)
  NEXT i
END SUB

' Moltiplicazione Matriciale P2=P1*S:
SUB s.transform(sx,sy) STATIC
  SHARED i
  x.punkte(i) = sx*x.punkte(i)
  y.punkte(i) = sy*y.punkte(i)
END SUB

' Dati dell'immagine:
' Numero dei punti / Numero delle linee:
DATA 33
DATA 30

' Coordinate del punto:
DATA  0, 3,    0, 7,    2, 8,    4, 8,    4,12
DATA  6,13,   20,13,   22,12,   22, 8,   24, 8
DATA 26, 7,   24,12,   26, 3,   24, 3,   24, 0
DATA 20, 0,   20, 3,    6, 3,    6, 0,    2, 0
DATA  2, 3,    6, 9,    6,12,   20,12,   20, 9
DATA 22, 5,   20, 6,   22, 7,   24, 6,    4, 5
DATA  2, 6,    4, 7,    6, 6
```

```
' Linee (Collegamenti dei punti):
DATA  0, 1,   1, 2,   2, 9,   9,10,   9,11
DATA 10,12,  12, 0,   3, 4,   4, 5,   5, 6
DATA  6, 7,   7, 8,  21,22,  22,23,  23,24
DATA 24,21,  13,14,  14,15,  15,16,  17,18
DATA 18,19,  19,20,  25,26,  26,27,  27,28
DATA 28,25,  29,30,  30,31,  31,32,  32,29
```

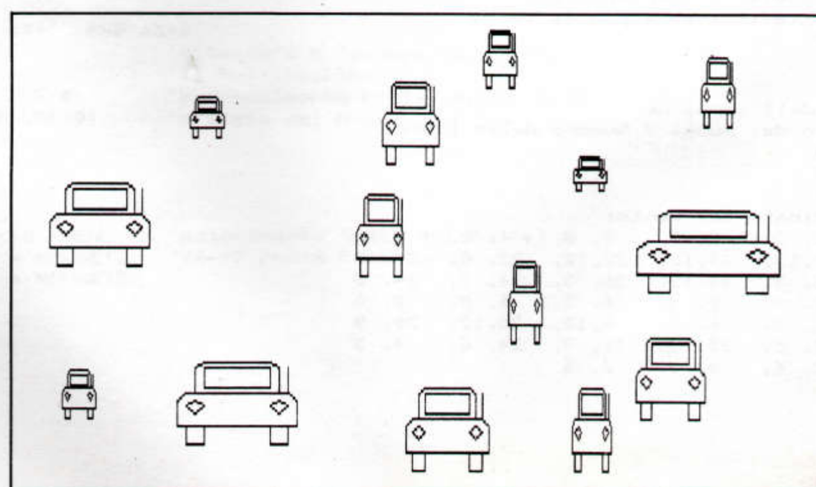


Fig. 3.9 Scala

In questo programma vengono già presentati gli elementi di base di una gestione grafica dei dati, nella quale ci addentreremo esaurientemente solo nel prossimo capitolo, al momento del primo incontro del "mondo" tridimensionale. Questo programma traccia una piccola immagine sul monitor. Tramite i tasti del cursore siamo ora in grado di ingrandire o ridurre l'immagine in direzione x oppure y. Le funzioni di tali tasti sono le seguenti:

Cursore verso l'alto	Ingrandimento in direzione y
Cursore verso il basso	Riduzione in direzione y
Cursore verso destra	Ingrandimento in direzione x
Cursore verso sinistra	Riduzione in direzione x
Esc	Termine del programma
Tasto del Mouse	Posizionamento dell'immagine a seconda della posizione del Mouse

La definizione dell'oggetto si trova nelle righe DATA alla fine del programma. In esse sono contenute le coordinate x ed y di ogni singolo vertice. Al termine sono contenuti i numeri di quei punti che descrivono un segmento. I dati vengono caricati in matrice e possono venire manipolati in qualunque momento come tale.

Le Sub-Routine **transformiere.bild ()** e **s.transform ()** sono molto importanti per il nostro compito. La seconda esegue una moltiplicazione matriciale di un singolo punto con la matrice di scala S (con gli elementi S_x e S_y). Il parametro "i" fornisce il numero del singolo punto nella matrice dei punti. L'altra routine trasforma tutti i punti dell'immagine, tutte le volte che viene chiamata da s.transform(), per ogni singolo punto.

Gli elementi matriciali S_x ed S_y provengono dal programma principale e vengono impostati a seconda delle immissioni che vengono effettuate tramite i tasti cursore. Il resto del programma è praticamente solo decorativo o complementare, e potrà venire modificato o ampliato in qualunque momento. Eventuali modifiche o ampliamenti sarebbero comunque il sistema migliore per prendere dimestichezza con l'argomento di questo programma.

3.2.3 Riflessioni

Grazie alla moltiplicazione delle matrici, anche le riflessioni attorno ad un determinato asse sono facilmente realizzabili. In questo caso è inoltre molto semplice trovare una matrice di trasformazione, dal momento che per la riflessione utilizzeremo le seguenti equazioni:

Specchiatura attorno all'asse x:

$$x' = x$$
$$y' = -y$$

Specchiatura attorno all'asse y:

$x' = -x$
$y' = y$

Ciò ci riporta alla scala, per cui abbiamo già la matrice adeguata:

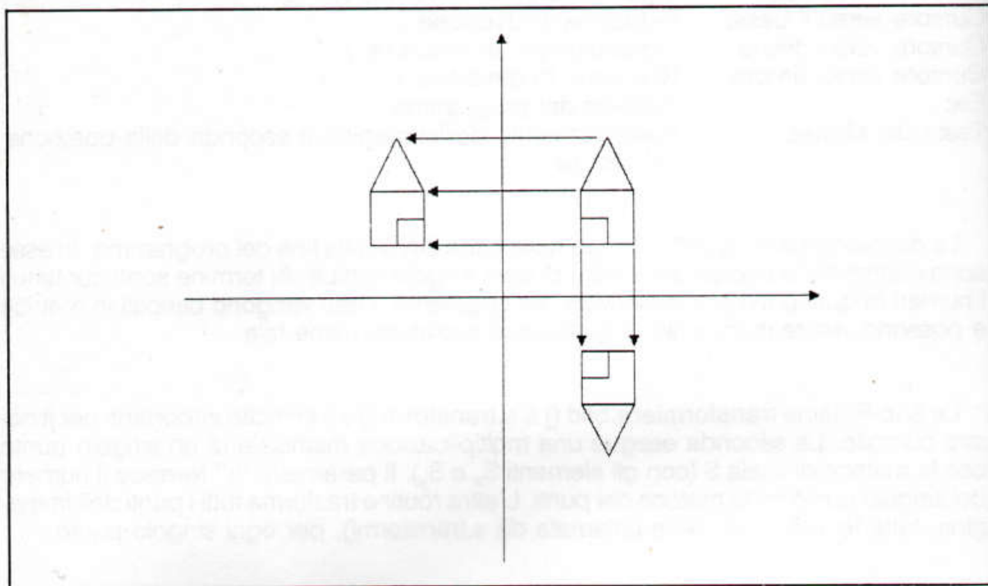


Fig. 3.10 Specchiatura

Specchiatura attorno all'asse x:

$$M_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Specchiatura attorno all'asse y:

$$M_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

Se ora vogliamo effettuare una riflessione attorno ad ambedue gli assi, cosa che corrisponderebbe ad una riflessione di punti, avremmo bisogno di eseguire solo una piccola moltiplicazione di matrici:

$$M_{xy} = M_x{}^*M_y = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

Dal momento che anche in questo caso si tratta dello stesso principio come per la messa in scala, non addentriamoci ulteriormente e passiamo all'argomento successivo. Per l'argomento presente, sarà sufficiente incorporare la riflessione nel programma precedentemente visto (es: riflessione in seguito alla pressione di un tasto, ecc).

3.2.4 Rotazioni bidimensionali

Stiamo per affrontare un argomento molto interessante, cioè la rotazione di singoli punti attorno all'origine delle coordinate. Spesso sarà infatti necessario ruotare singoli oggetti oppure intere immagini, al fine di poterle osservare da un altro punto di vista. L'elemento di base per una rotazione di questo genere attorno ad un punto a piacere è costituito dalla rotazione di un punto attorno al punto 0 delle coordinate, vediamo quindi di approfondire quest'ultimo aspetto.

Naturalmente ci interessa una matrice, con la quale sia possibile eseguire una rotazione di questo genere. Al fine di trovare tale matrice, dovremo dapprima fare una deviazione sulle equazioni di parametri corrispondenti. Osserviamo quindi l'immagine seguente:

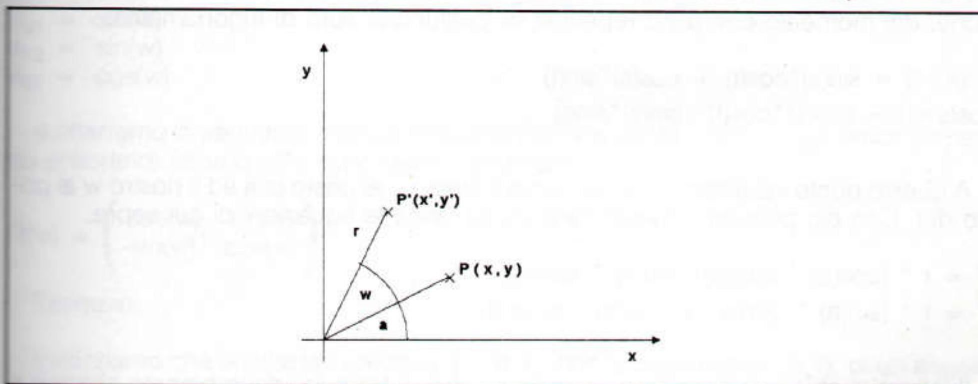


Fig. 3.11 Rotazione di un punto attorno all'origine

Forse non sappiamo, o abbiamo dimenticato, a cosa servano le funzioni angolari (seno o coseno, ecc). Nessun problema, possiamo rinfrescarci la memoria in appendice. In essa sono contenuti tutti gli elementi di base, e le spiegazioni di cosa significhi "funzioni angolari".

Per quanto concerne la figura, vediamo che in essa un punto P(x,y) viene ruotato dell'angolo w attorno al punto 0. Il punto risultante si chiama P'(x',y'). Sia P che P' hanno naturalmente la stessa distanza r dal centro di rotazione. Il collegamento tra P ed il punto 0 forma, insieme con l'asse x, l'angolo a. Di conseguenza, il collegamento fra P' ed il punto 0, rispetto all'asse x, forma l'angolo w+a.

Conformemente alle definizioni per seno e coseno, dal nostro schizzo potremo dedurre le seguenti quattro equazioni:

$$\cos(a) = x/r$$
$$\sin(a) = y/r$$
$$\cos(a+w) = x'/r$$
$$\sin(a+w) = y'/r$$

Con ciò avremmo già costituito un solido elemento di base, sul quale poter in seguito costruire. Ambedue gli ultimi rapporti possono venire trasformati con facilità in:

$$x' = r * \cos(a+w)$$
$$y' = r * \sin(a+w)$$

x' ed y', come noto, sono le due coordinate del punto ruotato, che noi stiamo cercando. x ed y rappresentano le coordinate del punto originale e w è l'angolo di rotazione.

E' probabile che i valori r ed a ci procurino un po' di confusione, dal momento che sono non noti. Cerchiamo quindi di sostituirli e per fare ciò utilizzeremo due cosidetti teoremi di addizione per funzioni angolari, per i quali non è necessaria una dimostrazione, dal momento che sono reperibili su qualunque libro di trigonometria:

$$\sin(s+t) = \sin(s)*\cos(t) + \cos(s)*\sin(t)$$
$$\cos(s+t) = \cos(s)*\cos(t) - \sin(s)*\sin(t)$$

A questo punto inseriremo naturalmente il nostro a al posto di s ed il nostro w al posto di t. Con ciò possiamo trasformare senza fatica le equazioni di cui sopra:

$$x' = r * [\cos(a) * \cos(w) - \sin(a) * \sin(w)]$$
$$y' = r * [\sin(a) * \cos(w) + \cos(a) * \sin(w)]$$

Il lettore si chiederà cosa fare con tale mostro di formula, dal momento che a e r sono ancora presenti. Osserviamo comunque le due ultimissime formule che non abbiamo ancora utilizzato. Esse ci permettono di sostituire tutti i sin(a) con y/r e tutti i cos(a) con x/r. Con ciò avremmo sostituito tutti i termini, nei quali era presente a:

$$x' = r * [x/r * \cos(w) - y/r * \sin(w)]$$
$$y' = r * [y/r * \cos(w) + x/r * \sin(w)]$$

A questo punto abbiamo ancora una sola grandezza non nota: r. Ma anche questa è solo apparente. Infatti, se eliminiamo le parentesi quadre, questo elemento di disturbo scomparirà immediatamente:

$$x' = x * \cos(w) - y * \sin(w)$$
$$y' = x * \sin(w) + y * \cos(w)$$

Abbiamo finalmente ottenuto ciò che volevamo: equazioni, dalle quali sia possibile calcolare, dai parametri noti x e y (coordinate del punto) e w (angolo di rotazione), le coordinate x' ed y' del punto ruotato.

A questo punto possiamo partire alla ricerca di una matrice adeguata. Osserviamo ancora una volta la moltiplicazione generica di una matrice di punti per una matrice quadrata, che era:

$$P' = P*M \quad = (x\ y) * \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$
$$= (x*a_{11} + y*a_{21} \quad x*a_{12} + y*a_{22})$$

Nel caso della rotazione, la matrice di punti risultante dovrebbe essere la seguente:

$$P' = (x*\cos(w) - y*\sin(w) \quad x*\sin(w) + y*\cos(w))$$

Ipotizziamo la seguente uguaglianza:

$a_{11} = \cos(w)$
$a_{21} = -\sin(w)$
$a_{12} = \sin(w)$
$a_{22} = \cos(w)$

e otterremo la seguente matrice di trasformazione per la rotazione generica (in senso antiorario) attorno all'origine delle coordinate:

$$R(w) = \begin{pmatrix} \cos(w) & \sin(w) \\ -\sin(w) & \cos(w) \end{pmatrix}$$

Esempio:

Ipotizziamo che si desideri ruotare il punto P, con le coordinate P(3,4), di un angolo di 60° attorno al punto 0. (Chi esegue il conteggio con la calcolatrice la dovrà impostare a DEG per un calcolo normale dei gradi. Normalmente i computer effettuano il calcolo con RAD). La nostra matrice di rotazione sarà quindi:

$$R(60) = \begin{pmatrix} \cos(60) & \sin(60) \\ -\sin(60) & \cos(60) \end{pmatrix} = \begin{pmatrix} 0.5 & 0.866 \\ -0.866 & 0.5 \end{pmatrix}$$

Il risultante punto P' potrà quindi venire ottenuto come segue:

$$P' = P(3,4) * R(60) = (3\ \ 4) * \begin{pmatrix} 0.5 & 0.866 \\ -0.866 & 0.5 \end{pmatrix}$$
$$= (3*0.5-4*0.866 \quad 3*0.866+4*0.5)$$
$$= (-1.964 \quad 4.598)$$

Le coordinate del punto risultante saranno quindi P'(-1.964, 4.598).

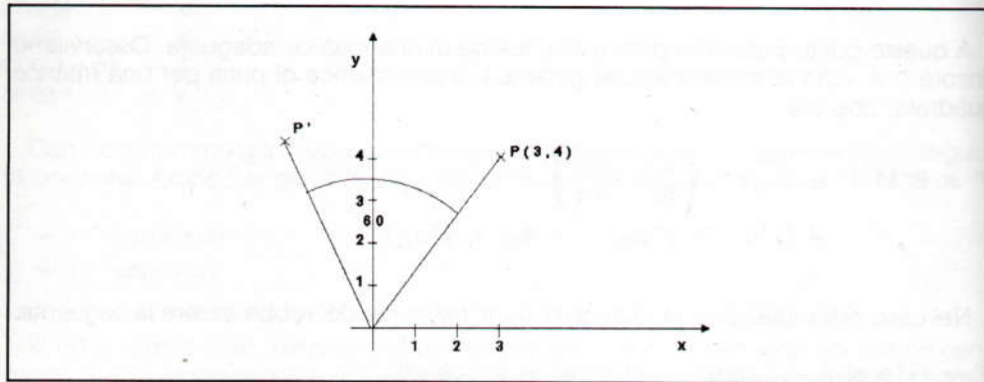


Fig. 3.12 Rotazione del punto P(3,4) di w=60 Gradi

A questo punto sono naturalmente molto semplici anche le trasformazioni combinate. Ipotizziamo di voler ruotare e ingrandire l'oggetto. Normalmente dovremmo effettuare queste due trasformazioni l'una dopo l'altra secondo la formula

$$P' = (P*R)*S$$

Dal momento però che, come abbiamo visto precedentemente, vale:

$$P' = (P*R)*S = P*(R*S)$$

potremo anche calcolare una sola matrice di trasformazione comune $M = R(w)*S(S_x,S_y)$, che esegua sia la rotazione che la scala:

$$M = R(w)*S(S_x,S_y) = \begin{pmatrix} \cos(w) & \sin(w) \\ -\sin(w) & \cos(w) \end{pmatrix} * \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$
$$= \begin{pmatrix} \cos(w)*S_x & \sin(w)*S_y \\ -\sin(w)*S_x & \cos(w)*S_y \end{pmatrix}$$

Nella stessa maniera, sarà possibile eseguire più rotazioni una dopo l'altra con una sola matrice.

Il seguente programma chiarirà ulteriormente il principio della rotazione:

```
'*************************************
'**                                 **
'**  Rotazione di un tratto di curva **
'**                                 **
'*************************************

'Dichiarazione Funzione:
DEF FNf(x)=SIN(x)/x

ON ERROR GOTO errore

'Parametri di funzione:
a = 320               'Coordinata x dell'origine
b = 150               'Coordinata y dell'origine
f1 = 10               'Fattore di ingrandimento x
f2 = 50               'Fattore di ingrandimento y

'Loop per modificare l'angolo di rotazione
FOR wi=0 TO 30 STEP 5
  w = wi/180 * 3.141593        'Convertire l'angolo in RAD

  'Inserimento dell'asse y delle coordinate:
  x = a : y = 200 : CALL rot.transform(w)
  PSET (xr%,yr%),2
  x = a : y =   0 : CALL rot.transform(w)
  LINE -(xr%,yr%),2

  'Inserimento dell'asse x delle coordinate:
  x =   0 : y = b : CALL rot.transform(w)
  PSET (xr%,yr%),2
  x = 640 : y = b : CALL rot.transform(w)
  LINE -(xr%,yr%),2

  x = 0
  y = b - f2*FNf((x-a)/f1)     'Calcolo del valore di funzione per x=0
  CALL rot.transform(w)
  PSET (xr%,yr%),3             'Impostazione del punto

  'Calcolo dei valori di funzione:
  FOR x=0 TO 639
    y = b - f2*FNf((x-a)/f1)   'Calcolo dei valori di funzione
    CALL rot.transform(w)      'e rotazione
    LINE -(xr%,yr%),3          'Linea dall'ultimo punto
    onerr:
  NEXT x
NEXT wi

END
```

```
'Trasformazione della rotazione per un punto:
SUB rot.transform(w) STATIC
  SHARED x,y,xr%,yr%
  xr% =  x*COS(w) + y*SIN(w)
  yr% = -x*SIN(w) + y*COS(w)
END SUB


errore:
  e=ERR
  IF e=11 OR e=6 THEN
    RESUME onerr
  END IF
ERROR e
```

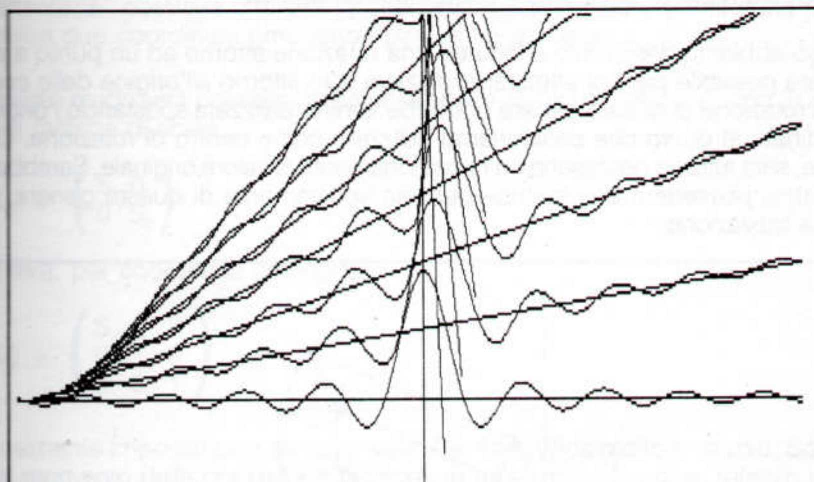


Fig. 3.13 Rotazione di un tratto di curva

Il programma presenta un'applicazione interessante della rotazione. In esso infatti la curva di una funzione di un determinato angolo w viene ruotata e tracciata sulla finestra di output che era stata definita nella riga DEF-FN (in questo programma f(x) = sin(x)/x). Il centro di rotazione si trova come sempre nel punto 0 delle coordinate della finestra grafica, dal momento che non è possibile effettuare delle rotazioni attorno ad un punto a piacere.

La routine che si occupa della rotazione si chiama **rot.transform()** e esegue con le coordinate x e y una rotaziona attorno all'angolo w che è stato fornito. Le coordinate risultanti vengono fornite da questo sotto-programma in xr e yr.

Riassumendo, il loop più esterno FOR...NEXT incrementa gradualmente la variabile wi dell'angolo di rotazione. Gli assi delle coordinate vengono tracciati brevemente (anch'essi ruotati) fornendo quindi il disegno della curva ruotata. In essa viene sempre tracciata una riga dall'ultimo punto tracciato della curva fino al punto attuale. Il primo punto per x = 0 dovrà di conseguenza venire calcolato separatamente. Cerchiamo di non farci irritare dalla formula un po' strana relativa al calcolo del valore della funzione. I parametri a, b, f1 e f2 servono solo a porre la funzione, correttamente e con dimensione esatta, all'interno della finestra (infatti anche in questo caso non si tratta di altro che di una trasformazione).

Per motivi di velocità può essere un vantaggio, in alcune applicazioni, utilizzare delle tabelle di seno e di coseno, invece di calcolare ogni volta queste funzioni angolari. Se si cerca il seno di un determinato angolo, sarà sufficiente che il programma faccia riferimento a tale tabella.

3.2.5 Spostamenti (traslazioni) e coordinate omogenee

Spesso abbiamo desiderato effettuare una rotazione attorno ad un punto a piacere (finora era possibile per noi effettuare rotazioni solo attorno all'origine delle coordinate). Una rotazione di questo genere potrebbe venire realizzata spostando l'origine delle coordinate al punto che desideriamo utilizzare come centro di rotazione. Dopo la rotazione, sarà tuttavia necessario un riposizionamento al valore originale. Sarebbe inoltre desiderabile possedere una matrice per uno spostamento di questo genere, cioè la cosidetta traslazione.

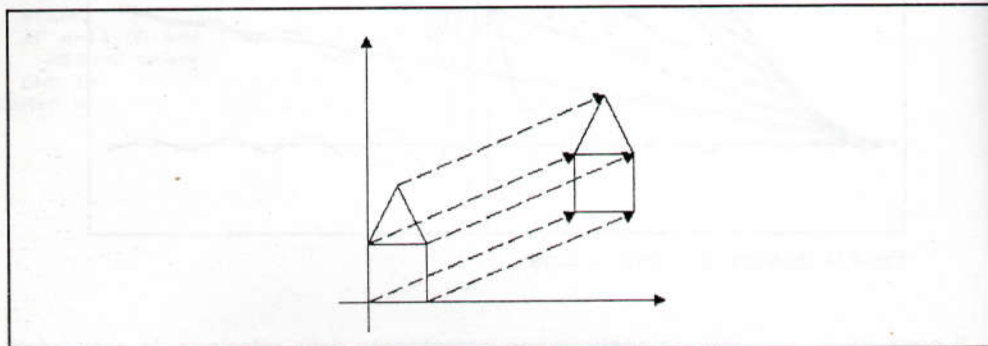


Fig. 3.14 Traslazione (Spostamento)

Lo spostamento di un punto è molto semplice in sè e per sè:

$x' = x + T_x$
$y' = y + T_y$

dove T_x e T_y rappresentano i valori dello spostamento in direzione x e y. Purtroppo per queste equazioni non è possibile creare una matrice di trasformazione 2x2. Per questo motivo sono state introdotte dai matematici le cosidette coordinate omogenee. Al posto delle nostre vecchie matrici 2x2, in futuro effettueremo calcoli solo con matrici di trasformazione 3x3.

Per quanto corncerne le coordinate omogenee, in esse le coordinate di un punto non vengono più indicate con due valori, come era usuale finora. Ne utilizzeremo invece tre. La nostra matrice di punti sarebbe quindi:

$P = (x*n \; y*n \; n)$

Il valore n è per così dire una coordinata Dummy, cioè una coordinata che non esiste nella realtà e che viene introdotta solo per motivi di calcolo. Vedremo in seguito come questa coordinata addizionale venga eliminata automaticamente, non appena verrà eseguita una trasformazione. L'introduzione di coordinate omogenee è molto importante, non solo per questo problema. Anche nel capitolo seguente ritorneremo spesso su questo argomento.

Importante: E' possibile ottenere le coordinate esatte x ed y tramite una divisione delle prime due coordinate omogenee per un valore terzo n.

Allo stesso modo sarà possibile ampliare le nostre matrici di trasformazione 2x2. Quindi anche la matrice di scala

$$S(S_x, S_y) = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

diventerà, per coordinate omogenee

$$S(S_x, S_y) = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Normalmente imposteremo sempre n = 1. Ciò semplifica molto il calcolo. Solo in seguito, al momento della cosidetta proiezione in prospettiva (capitolo relativo alla tridimensione) ritorneremo su questo argomento. Una trasformazione di un punto potrà di conseguenza venire eseguita come segue:

$$P' = P(x,y) * S(S_x,S_y)$$

$$= (x{*}n \; y{*}n \; n) * \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = (S_x{*}x{*}n \; S_y{*}y{*} \; n \; n)$$

Tramite la divisione per n otterremo le coordinate normali:

$$P' = (S_x{*}x \; S_y{*}y)$$

Ciò è esattamente il risultato che conoscevamo già. Dal momento che n = 1, questo calcolo sarà spesso molto più semplice di quanto non appaia qui.

Possiamo trasformare allo stesso modo tutte le altre matrici di trasformazione (matrici omogenee), ampliandole semplicemente con "quattro 0 ed un 1".

$$R(w) = \begin{pmatrix} \cos(w) & \sin(w) & 0 \\ -\sin(w) & \cos(w) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Il risultato resta comunque lo stesso che conoscevamo già.

A questo punto ci interesserà conoscere come deve essere una matrice di trasformazione per una traslazione. Eccola:

$$T(T_x,T_y) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{pmatrix}$$

In essa, T_x e T_y (come sopra) sono i valori di spostamento in direzione x ed y. Moltiplichiamo ora un punto concreto (es: P(5,9)) con una matrice di traslazione completa. Vedrete che funziona! A questo punto abbiamo posto le basi per molte elaborazioni grafiche, che passiamo ad esaminare immediatamente nel paragrafo seguente.

3.2.6 Rotazione attorno ad un punto a piacere

La soluzione di questo problema dovrebbe esserci già nota (l'abbiamo delineata brevemente poco fa). Al fine di far ruotare un oggetto, o meglio un punto, attorno ad un punto a piacere su di un piano, sono necessarie tre trasformazioni singole: dapprima uno spostamento dell'origine delle coordinate fino al punto che abbiamo scelto come centro di rotazione (in altre parole: uno spostamento dell'immagine con il centro di rotazione al punto 0); quindi una rotazione attorno all'origine delle coordinate (e con ciò attorno a tale centro di rotazione); ed infine un riposizionamento dell'immagine alla posizione originale.

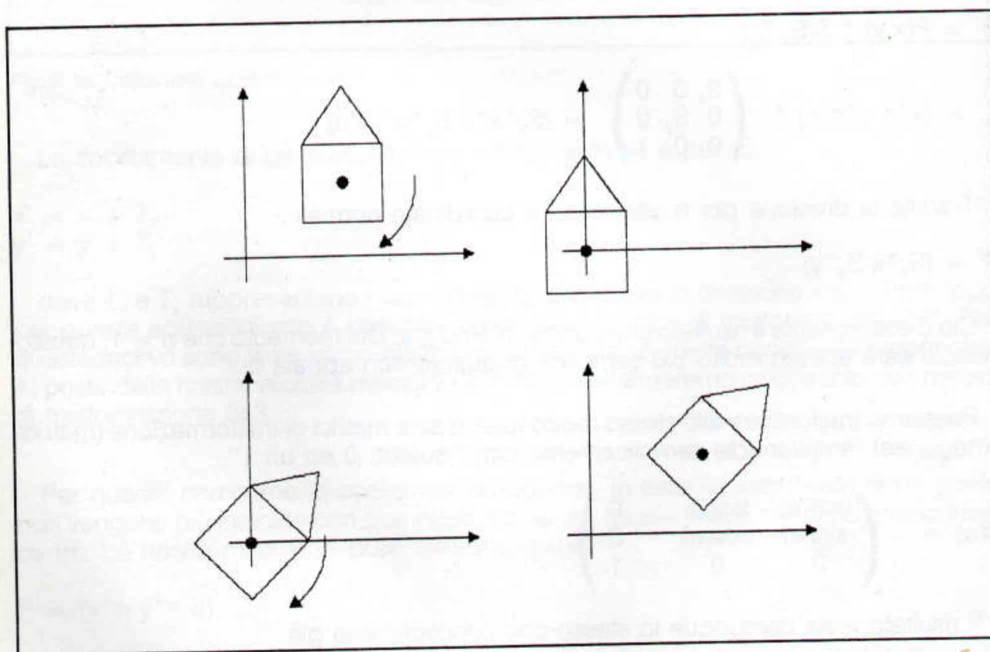


Fig. 3.15 Rotazione attorno ad un punto arbitrario (traslazione, rotazione e nuova traslazione)

Quindi si tratta di una traslazione, di una rotazione e di una ulteriore traslazione finale:

$$P' = [(P*T_1)*R]*T_2 = P*(T_1*R*T_2)$$

Ipotizzando che il centro della rotazione debba essere: $Z(x_z, y_z)$, la matrice di traslazione che sposta Z al punto di origine, sarà:

$$T_1(-x_z,-y_z) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_z & -y_z & 1 \end{pmatrix}$$

La matrice di rotazione, come è noto, sarà:

$$R(w) = \begin{pmatrix} \cos(w) & \sin(w) & 0 \\ -\sin(w) & \cos(w) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

e la matrice che riposiziona il centro di rotazione, sarà:

$$T_2(-x_z,-y_z) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_z & y_z & 1 \end{pmatrix}$$

A questo punto dovremo solo determinare il prodotto $T_1 * R * T_2$ di queste tre matrici, ed otterremo:

$$R'(w,x_z,y_z) = T_1(-x_z,-y_z) * R(w) * T_2(x_z,y_z) =$$

$$\begin{pmatrix} \cos(w) & \sin(w) & 0 \\ -\sin(w) & \cos(w) & 0 \\ -x_z*\cos(w)+y_z*\sin(w)+x_z & -x_z*\sin(w)-y_z*\cos(w)+y_z & 1 \end{pmatrix}$$

La notazione sotto forma di parametri di una rotazione di un punto P(x,y) sarà quindi la seguente:

$$x' = x*\cos(w) - y*\sin(w) - x_z*\cos(w) + y_z*\sin(w) + x_z$$
$$y' = x*\sin(w) + y*\cos(w) - x_z*\sin(w) - y_z*\cos(w) + y_z$$

La matrice di rotazione R'(w) per le rotazioni attorno a punti a piacere, risulterà forse un po' complicata, ma dimostrerà sicuramente e in maniera convincente che non è stata assolutamente una cattiva idea quella di introdurre il calcolo matriciale. In tal modo siamo quindi in grado di riassumere qualunque combinazione di trasformazioni a piacere in una sola matrice, e di programmare tutto ciò in maniera compatta ed efficiente in un programma. Con le matrici di trasformazione, a dire il vero, è possibile riassumere tutto ciò che si può voler fare con un punto, quindi anche una rotazione a spirale oppure il suo posizionamento su di un'altra funzione, come vedremo meglio in seguito al termine di questa. Per quanto riguarda la notazione in formato matriciale, forse c'è qualcosa che salta agli occhi nella matrice R' di cui sopra. Essa può anche venire sostituita da una rotazione con traslazione finale (quindi senza traslazione iniziale). La matrice di traslazione conterrebbe quindi i valori della terza riga di R'. Per una migliore comprensione e naturalmente per verificarne il funzionamento, applichiamo quanto appena appreso in un piccolo programma:

```
'***************************
'**                       **
'**  Rotazione attorno ad **
'**   un punto a scelta   **
'**                       **
'***************************

'Coordinate dell' oggetto della rotazione:
'(Rettangolo)
xr(0) = 100 : yr(0) = 100
xr(1) = 100 : yr(1) = 120
xr(2) = 160 : yr(2) = 120
xr(3) = 160 : yr(3) = 100

' Costante Pi:
pi = 3.141593

' Coordinate iniziali del centro di rotazione:
xz = 130
yz = 110

' Angolo di rotazione:
w = pi/20

' Loop principale:
WHILE INKEY$=""                        'Attende la pressione di un tasto

  IF MOUSE(0)=1 THEN                   'Tasto mouse premuto ?
    xz = MOUSE(3)                      'si -> Coordinate del mouse
    yz = MOUSE(4)                      'come nuovo centro di rotazione
  END IF

  'Trasformazione di 4 coordinate angolari:
  FOR i=0 TO 3
    x = xr(i) : y = yr(i)
  CALL rot2.transform(w,xz,yz)
    xr(i) = xr : yr(i) = yr
  NEXT i

  CLS                                  'Cancella schermo
  PSET (xz,yz),1                       'Tracciatura centro di rotazione
  LINE (xr(0),yr(0))-(xr(1),yr(1)),3
  LINE -(xr(2),yr(2)),3                'Tracciatura rettangolo
  LINE -(xr(3),yr(3)),3
  LINE -(xr(0),yr(0)),3
WEND
```

```
' Trasformazione (Rotazione attorno ad un punto a piac
' w  - Angolo di rotazione
' xz - Coordinata x del centro di rotazione
' yz - Coordinata y del centro di rotazione

SUB rot2.transform(w,xz,yz) STATIC
  SHARED x,y,xr,yr

  si = SIN(w)                        'Calcolo costanti
  co = COS(w)
  xr = x*co-y*si-xz*co+yz*si+xz
  yr = x*si+y*co-xz*si-yz*co+yz
END SUB
```

Non appena avremo lanciato questo piccolo programma, un rettangolo comincerà a ruotare. Nella finestra di output un piccolo punto indica il centro di rotazione. Prendiamo quindi il mouse, posizioniamo il puntatore su di un punto a piacere all'interno della finestra e azioniamo il pulsante sinistro. Il centro di rotazione cambierà immediatamente, diventando il punto sul quale abbiamo clickato. Se si vuole abbandonare il programma, sarà sufficiente una singola pressione di un tasto qualunque.

Per quanto concerne le funzioni, vediamo che all'inizio della routine, i quattro vertici del rettangolo da ruotare vengono definiti in due matrici (xr() e yr()), mentre il centro di rotazione viene definito in x_z ed y_z e l'angolo di rotazione viene definito in w. Dopo tali righe troviamo il loop principale, che verrà abbandonato non appena si preme un tasto a piacere. La funzione **MOUSE(0)** segnala un evento del mouse (ved. manuale del Basic), che verrà utilizzato per inserire nuove coordinate per il centro di rotazione.

La trasformazione vera e propria dei quattro vertici ha luogo in un piccolo loop FOR...NEXT il quale a sua volta chiama il sottoprogramma **rot2.transform()** per la moltiplicazione delle matrici. Le costanti si e co contengono i valori di seno e coseno di w. In verità, anche tali valori avrebbero dovuto venire definiti all'inizio del programma (risparmiando anche tempo di calcolo) dal momento che l'angolo non viene mai modificato. Abbiamo evitato di fare ciò per rendere maggiormente comprensibile il programma stesso.

Una volta determinati i quattro punti ruotati, il programma cancella il contenuto della finestra di output ed inizia il lavoro di tracciatura.

3.2.7 Una piccola leccornia: deformazione di aree video

Per la comprensione delle operazioni bi e tridimensionali, il seguente capitolo non è assolutamente necessario (i lettori che vanno di fretta potrebbero anche saltarlo), ma ci permette di familiarizzare con un aspetto secondario anch'esso interessante.

Si tratta di qualcosa che forse abbiamo già visto: un programma proietta un intero blocco grafico lungo una curva sinusoidale deformandolo. Vediamo subito se si tratta di un procedimento semplice oppure no. Anche in questo caso abbiamo a che fare di nuovo con operazioni, rappresentabili sotto forma di matrici.

Di solito si calcolano i valori di una normale funzione (esempio la funzione seno) e li si somma alle coordinate x ed y di ogni singolo punto di un blocco grafico (quasi sempre rettangolare). Sotto forma di parametri, è possibile realizzare ciò nella maniera seguente:

$x' = x + f(y)$
$y' = y + g(x)$

e sotto forma di matrici:

$$P' = P * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ f(y) & g(x) & 0 \end{pmatrix}$$

Invece di f(y) e g(x) avremmo potuto scrivere senza problemi f(x) e g(y). Vediamo ora un semplice esempio: la proiezione di un blocco grafico lungo una curva sinusoidale.

A questo punto ci scontriamo con un altro problema: ipotizziamo che il blocco grafico rettangolare che vogliamo proiettare si trovi già su un punto qualunque dello schermo (o in memoria). Se lo vogliamo deformare, dobbiamo controllare ogni singolo punto di questo blocco per poi trasformarlo (deformarlo) secondo quanto desideriamo. Per fare ciò useremo in Basic il comando **POINT**, che ci fornisce il colore di un punto sullo schermo. Il seguente programma chiarirà meglio questo aspetto:

```
'*************************
'**                    **
'**   Trasformazione di **
'**   Blocchi schermo   **
'**                    **
'*************************

pi = 3.141593

'Definizione del campo da trasformare:

'Definizione di un nuovo motivo di riempimento:
DIM muster%(7)
muster%(0) = &H0
muster%(1) = &H7444
muster%(2) = &H4444
muster%(3) = &H6444
muster%(4) = &H4444
muster%(5) = &H7774
PATTERN &HFFFF,muster%
```

```
LINE (0,0)-(300,40),3,BF
PRINT : PRINT
PRINT TAB(4) "C o d a   d i   m a c c h i n e"

'Indicazione dimensioni del blocco video da trasformare:
xq% = 0             'Coordinata x in alto a sinistra
yq% = 0             'Coordinata y in alto a sinistra
br% = 300           'Larghezza
ho% = 40            'Altezza

'Posizione del blocco di destinazione:
xz = 10
yz = 100

'Funzione della curva di trasformazione g(x):
DEF FNg(x) = ampl*SIN(anzs*x/(br%/(2*pi)))

'Ampiezza della curva di trasformazione:
ampl = 20

'Numero dei periodi di oscillazione per blocco:
anzs = 1

'Routine di disegno:
FOR x0=xq% TO br%-1              'Elaborazione
  FOR y0=yq% TO ho%-1            'punto per punto
    c% = POINT(x0,y0)            'Determinazione del colore del punto sorgente
    IF c%<0 THEN                 'se uguale a -1 => e' esterno alla finestra
      COLOR 1
      PRINT "Errore!"
      PRINT "Il campo sorgente non si trova"
      PRINT "completamente entro la finestra!!!"
      GOTO schluss
    END IF
    COLOR c%                     'Regolazione del colore
    x = x0-xq%                   'Calcolo delle coordinate relative sorgenti
    y = y0-yq%
    y = FNg(x)+y                 'e esecuzione della Trasformazione
    PSET (xz+x,yz+y)             'Posizionamento del punto di destinazione
  NEXT y0
NEXT x0

schluss:
COLOR 1
```

```
'Ri-normalizzazione del motivo:
DIM mu%(1)
mu%(0) = &HFFFF
mu%(1) = &HFFFF
PATTERN ,mu%
```

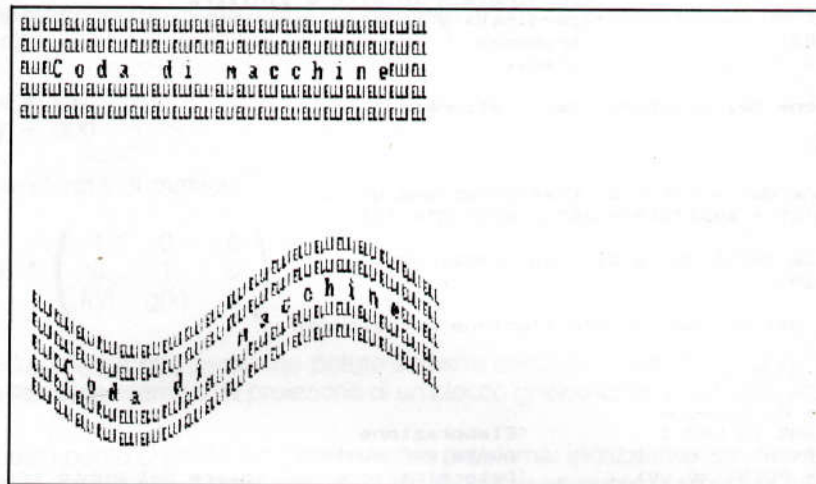


Fig. 3.16 Deformazione di porzioni di schermo

Dapprima viene costituito il campo sorgente da deformare. Un bordo predefinito ed un po' di testo renderanno più chiaro l'effetto. Vengono poi dichiarate le dimensioni del blocco da deformare e la posizione del blocco deformato che rimarranno costanti. Quindi viene definita la funzione di trasformazione.

Come abbiamo visto, sono state utilizzate per f(y) e g(x) le seguenti funzioni:

$$f(y) = 0$$

$$g(x) = ampl * \sin(x * \frac{anzs}{br/(2*pi)})$$

dove:

ampl = ampiezza della curva sinusoidale
anzs = numero delle oscillazioni in un blocco grafico
br = larghezza in punti del blocco grafico
x = coordinata x relativa (relativa allo 0 del blocco)

Alla coordinata x di ogni singolo punto scandito da POINT non viene aggiunto nulla. Alla coordinata y, invece, il programma aggiunge il valore della funzione di una curva sinusoidale. Il risultato sarà visibile sul vostro schermo. L'espressione

$$\frac{anzs}{br/(2*pi)}$$

rappresenta il fattore di espansione della curva sinusoidale. Il suo valore determina anche quanto deve essere lungo un periodo di oscillazione. Esso viene determinato in modo che l'utente possa indicare per quante oscillazioni il blocco grafico da deformare deve venire "modulato". br inoltre indica la larghezza di questo punto (quanti punti sono contenuti nel periodo).

Naturalmente, come sempre, è possibile modificare a piacere la funzione. Di solito non è molto difficile, come dimostrato dall'esempio seguente (è possibile anche prevedere delle lunghezze di oscillazione e delle ampiezze fisse). Proviamo a sostituire come esempio le seguenti righe prima di PSET:

```
x = x0-xq%              'calcolo coordinate relative
y = y0-yq%
y = FNg(x)+y            'ed esecuzione della trasformazione
```

con quanto segue:

```
xt = x0-xq%             'calcolo coordinate relative
yt = y0-yq%
ampl = 10               'Ampiezza per la coordinata x
br% = 40                'Altezza del blocco
x = FNg(yt)+xt          'Deformazione anche per coordinate x
ampl=20                 'Ampiezza per la coordinata y
br% = 300               'Larghezza del blocco
y = FNg(xt)+yt          'ed esecuzione della trasformazione
```

In questo caso vengono trasformate anche le coordinate x. Proviamo ad osservare il risultato: assume una dimensione quasi spaziale.

E' possibile provare anche con altre funzioni, ecco un paio di consigli:

L'equazione di un semi ellisse (vedi inizio del capitolo) trasformata sulla coordinata y, suscita l'impressione che il blocco grafico sia avvolto attorno a se stesso. In seguito potremo proiettare in prospettiva e ruotare nello spazio un blocco; o avvolgerlo addirittura su di una sfera od un cubo, come la carta decorata sui regali di Natale.

3.3 Clipping bidimensionale, cioè: finestre sulla grafica

Il lettore si sarà già chiesto come determinare le zone visibili di un cerchio, di una retta o di un rettangolo all'interno di una determinata finestra. Questo problema è noto come CLIPPING.

Normalmente non avremo molto a che fare con il **Clipping**. Quando lavoreremo con finestre (o Layers) il sistema operativo si occuperà di tutto. Ciononostante, in alcune applicazioni, siamo costretti ad occuparci noi di tale aspetto. Vediamo quindi le soluzioni per i punti e per le linee:

3.3.1 Clipping di punti

Per quanto riguarda un punto, la questione centrale è quella di sapere se esso può venire tracciato oppure no. La risposta non può essere altro che un sì oppure un no. Un punto non può venire suddiviso come una linea o un cerchio. Come fare quindi per verificare se un punto si trova all'interno o all'esterno di una finestra?

Per semplificare, attribuiamo i seguenti nomi alle coordinate di delimitazione di una finestra:

x_{min} coordinata x angolo superiore sinistro
y_{min} coordinata y angolo superiore sinistro
x_{max} coordinata x angolo inferiore destro
y_{max} coordinata y angolo inferiore destro

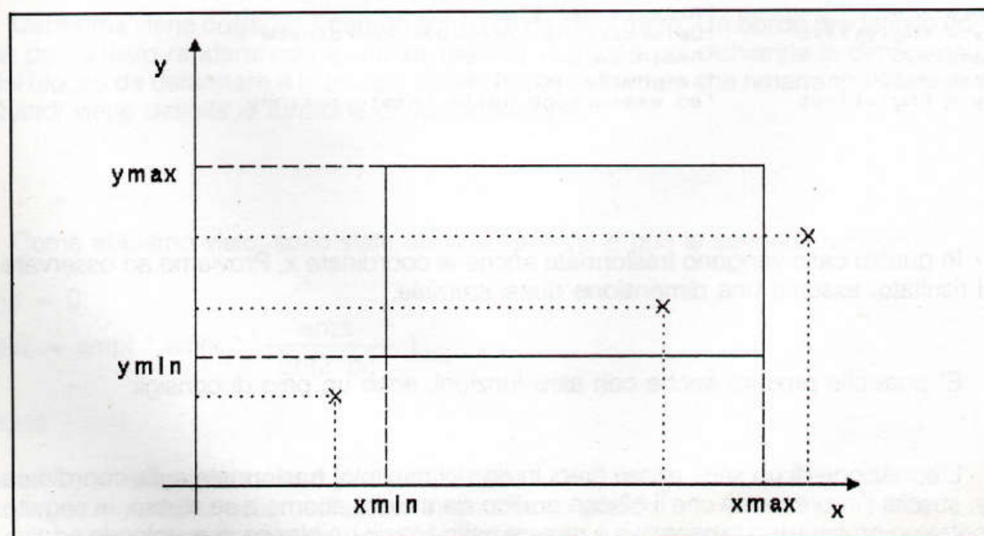


Fig. 3.17 Clipping di un punto

Un punto singolo avente le coordinate x ed y si troverà quindi all'interno di una finestra se tutte e quattro le seguenti uguaglianze sono vere:

$x \geqq x_{min}$ $\quad x \leqq x_{max}$
$y \geqq y_{min}$ $\quad y \leqq y_{max}$

Se anche una sola di queste quattro uguaglianze è falsa, il punto si troverà all'esterno della finestra e di conseguenza non sarà visibile. Naturalmente è possibile eseguire questo test per ogni punto di una figura ecc, ma non sarebbe molto efficiente.

3.3.2 Clipping di linee

Sono stati sviluppati dei procedimenti di calcolo molto veloci, al fine di effettuare il clipping di linee. L'algoritmo che svolge questa funzione ha assunto il nome dei suoi sviluppatori, Cohen e Sutherland, e viene presentato brevemente in questa sede.

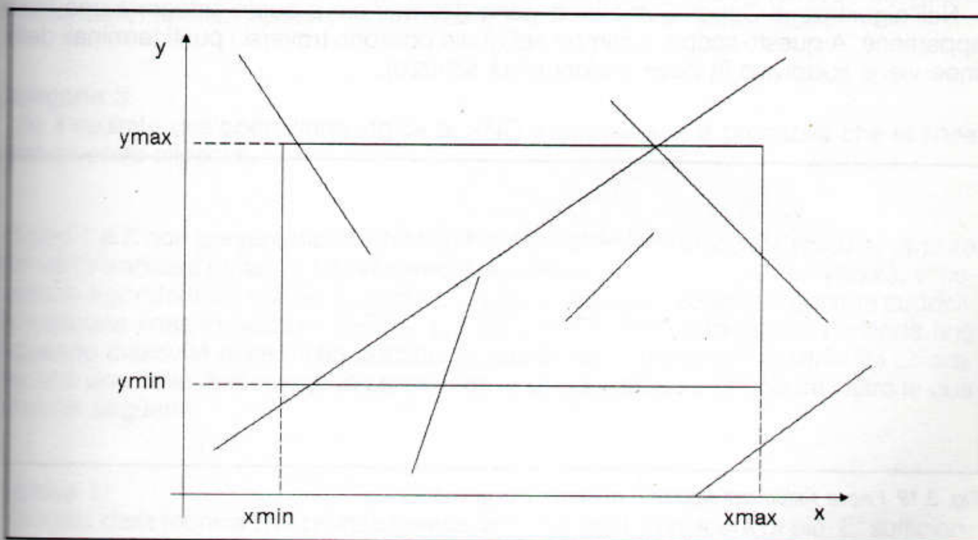


Fig. 3.18 Clipping di linee

L'algoritmo:
Esistono in linea di massima tre diverse categorie di linee (con il punto di partenza x_1, y_1 e con il punto di arrivo x_2, y_2), che possono venire elaborate:

Categoria 1:
Questa categoria comprende le linee che sono completamente visibili, cioè quelle i cui punti di partenza e di termine si trovano all'interno della finestra. Per ambedue i punti x_1, y_1 e x_2, y_2 valgono quindi le formule di cui sopra relative alla visibilità di un punto. E' ovvio che tali linee possono venire tracciate immediatamente.

Categoria 2:
Si tratta di linee che sono assolutamente invisibili in ogni caso. Ciò accade quando anche una sola delle seguenti espressioni è vera:

a) $x_1 > x_{max}$ e $x_2 > x_{max}$
b) $x_1 < x_{min}$ e $x_2 < x_{min}$
c) $y_1 > y_{max}$ e $y_2 > y_{max}$
d) $y_1 < y_{min}$ e $y_2 < y_{min}$

Non essendo visibili, tali linee non necessitano quindi di venire tracciate.

Categoria 3:
Questa categoria comprende le linee che non adempiono alle condizioni di cui sopra. Si tratta infatti di linee completamente invisibili oppure visibili solo parzialmente. E' questa categora di linee che ci creerà in seguito maggior confusione.

Nell'algoritmo di Cohen-Sutherland viene determinato a quale categoria una linea appartiene. A questo scopo, il campo nel quale possono trovarsi i punti terminali delle linee viene suddiviso in nove sezioni (vedi schizzo):

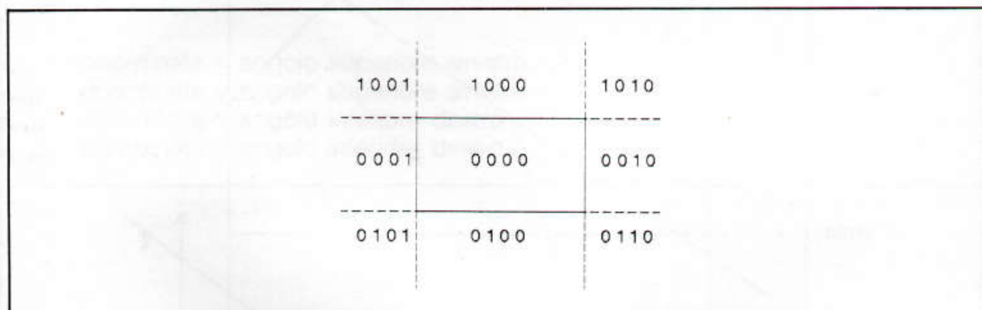


Fig. 3.19 I nove settori dell'algoritmo di Cohen-Sutherland

Per ogni punto finale della linea viene utilizzata una memoria di 4 bit. I singoli bit per un punto vengono quindi impostati secondo il seguente schema (Bit = 1) oppure azzerati (Bit = 0):

Bit 3	= 1 :	$y-y_{max}$	$\geqq$ 0	(Il punto si trova al di sopra della finestra)
	= 0 :	$y-y_{max}$	$<$ 0	(Il punto non si trova al di sopra della finestra)
Bit 2	= 1 :	$y_{min}-y$	$\geqq$ 0	(Il punto si trova sotto alla finestra)
	= 0 :	$y_{min}-y$	$<$ 0	(Il punto non si trova sotto alla finestra)
Bit 1	= 1 :	$x-x_{max}$	$\geqq$ 0	(Il punto si trova a destra della finestra)
	= 0 :	$x-x_{max}$	$<$ 0	(Il punto non si trova a destra della finestra)
Bit 0	= 1 :	$x_{min}-x$	$\geqq$ 0	(Il punto si trova a sinistra della finestra)
	= 0 :	$x_{min}-x$	$<$ 0	(Il punto non si trova a sinistra della finestra)

In questo caso il Bit 0 è il bit più a destra di tutti nella cella di memoria. Nel caso in cui per un punto, dopo questi quattro test, risulti un valore 0 per tutti e quattro i bit, il punto si troverà all'interno della finestra.

A questo punto è interessante vedere come riuscire a determinare l'appartenenza delle linee ad ogni categoria. Dovremo semplicemente collegare con una operazione logica di AND i valori dei quattro bit così ottenuti per gli estremi di una linea, per arrivare ai seguenti risultati:

Categoria 1:
Nel caso in cui ambedue i valori siano uguali a 0 (l'operazione logica di OR è 0), la linea apparterrà alla categoria 1, cioè sarà completamente visibile.

Categoria 2:
Nel caso in cui l'operazione logica di AND di ambedue i valori fornisca un valore diverso da 0, la linea apparterrà alla categoria 2, cioè non sarà visibile.

Categoria 3:
Se il risultato dell'operazione logica di AND è uguale a 0, è probabile che la linea debba venire clippata.

I casi 1 e 2 non presentano alcun problema: la linea verrà tracciata (caso 1) oppure non verrà tracciata (caso 2). La categora 3, al contrario, è un po' più complicata. E'necessario approfondire meglio questo tipo di linee. A questo scopo l'algoritmo suddivide ciascuna linea in porzioni sempre più piccole. Questa suddivisione continua fino a quando ciascuna delle molte porzioni di questa linea potrà venire attribuita chiaramente a una delle due categorie precedenti. A questo scopo esistono tra l'altro le due tecniche seguenti:

Tecnica 1:
Si tratta della tecnica che probabilmente, a prima vista, ci piacerà di più. E' sufficiente calcolare i punti di intersezione della linea da tagliare con i lati della finestra. Ciò naturalmente può richiedere molto tempo. Tuttavia, in caso di finestre rettangolari, nelle quali i quattro lati si trovino paralleli o perpendicolari agli assi delle coordinate, i test divengono ancora più semplici: é sufficiente utilizzare i codici dei 4 bit di ambedue i punti finali delle linee di cui sopra. Secondo i seguenti criteri è quindi possibile determinare con quali lati della finestra è necessario tagliare la linea (per una linea è possibile che si presenti un massimo di 4 casi contemporaneamente):

Caso 1:	bit 3 = 1:	Punto di intersezione con la linea	$y = y_{max}$
Caso 2:	bit 2 = 1:	Punto di intersezione con la linea	$y = y_{min}$
Caso 3:	bit 1 = 1:	Punto di intersezione con la linea	$x = x_{max}$
Caso 4:	bit 0 = 1:	Punto di intersezione con la linea	$x = x_{min}$

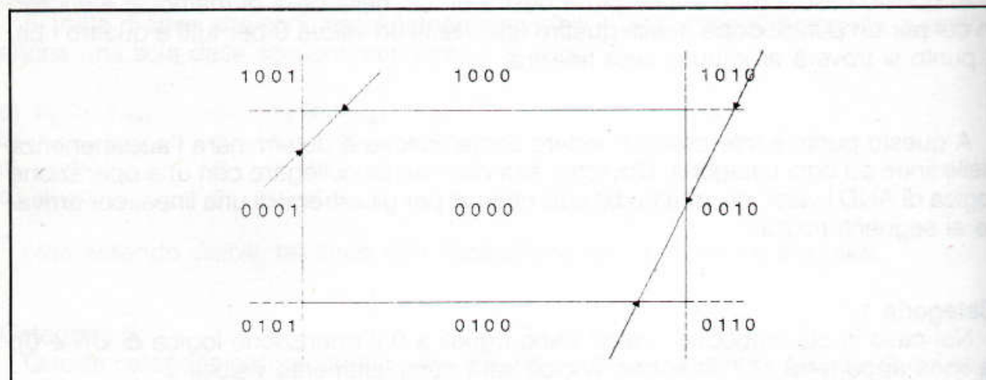


Fig. 3.20 Calcolo del punto di intersezione secondo Cohen-Sutherland

Dopo che si è determinato con quali lati si deve tagliare la linea, addentriamoci nelle formule secondo le quali è possibile determinare i punti di intersezione esatti (cioè i nuovi punti finali della linea). Il punto di partenza per fare ciò è l'equazione delle rette passanti per punti contenuta nell'appendice.

Caso 1 e 2:

Conosciamo già la coordinata y del punto di intersezione, cioè y_{max} (Caso 1) oppure y_{min} (caso 2). Ciò che resta da determinare è la coordinata x, ancora sconosciuta:

$$x = \frac{x_2 - x_1}{y_2 - y_1} * (y - y_2) + x_2$$

Casi 3 e 4:

In questo caso ci troviamo di fronte a rapporti invertiti. La coordinata x del punto di intersezione è nota (x_{max} per caso 3 e x_{min} per caso 4). Stiamo invece cercando la coordinata y: .

$$y = \frac{y_2 - y_1}{x_2 - x_1} * (x - x_2) + y_2$$

Per ambedue le formule vale:

x	coordinata x del punto di intersezione
y	coordinata y del punto di intersezione
x_1	coordinata x del punto di partenza della linea
y_1	coordinata y del punto di partenza della linea
x_2	coordinata x del punto finale della linea
y_2	coordinata y del punto finale della linea

Per ogni linea dovremo quindi calcolare da un minimo di 1 ad un massimo di 4 punti di intersezione, i quali suddividono la linea in porzioni che vanno da 2 a 5, delle quali però solo una può essere visibile. Tramite il calcolo dei 4 bit sarà possibile determinare di quale porzione di linea si tratta. I punti finali delle linee accorciate sono noti, quindi non perdiamo altro tempo e facciamo apparire sullo schermo la porzione determinata.

Tecnica 2:

Il procedimento di calcolo precedentemente descritto è perfettamente adeguato per dei rapporti normali. Tuttavia in Assembler e per alcuni problemi, che necessitano di venire elaborati molto velocemente, tale formula non è sufficiente. Al fine di poter determinare la porzione giusta della linea oppure i suoi punti finali, sono necessarie divisioni e moltiplicazioni decimali. Dal momento che ciò potrà sembrare piuttosto antipatico a molti lettori, indirizziamoci verso un metodo nel quale è possibile utilizzare senza eccezioni l'aritmentica intera. Anzi: saranno sufficienti addizioni intere, sottrazioni intere e istruzioni di shift (divisione per 2, 4 ecc).

L'algoritmo viene svolto in maniera ricorsiva. Suddividiamo dapprima la linea in due porzioni uguali. Il punto intermedio PM (x_m, y_m) di una linea può venire determinato tramite le seguenti formule, che sono facili da calcolare tramite semplici operazioni intere:

$$x_m = \frac{x_1 + x_2}{2} \qquad y_m = \frac{y_1 + y_2}{2}$$

A questo punto abbiamo due linee, per le quali effettueremo di nuovo la suddivisione in categorie (vedi sopra). Quelle porzioni di linea che rientrano nella categoria 3 verranno di nuovo dimezzate ecc. Ciò verrà ripetuto fino a quando ciascuna piccolissima porzione di linea potrà venire attribuita univocamente ad una delle prime due categorie. A questo punto sapremo quali linee potremo tracciare e quali no. Il presente algoritmo è particolarmente indicato per il liguaggio C, il quale supporta completamente le ricorsività. Tuttavia, anche in assembler 68000, con i comandi **LINK** e **UNLINK** sono possibili le variabili locali e di conseguenza la recursione.

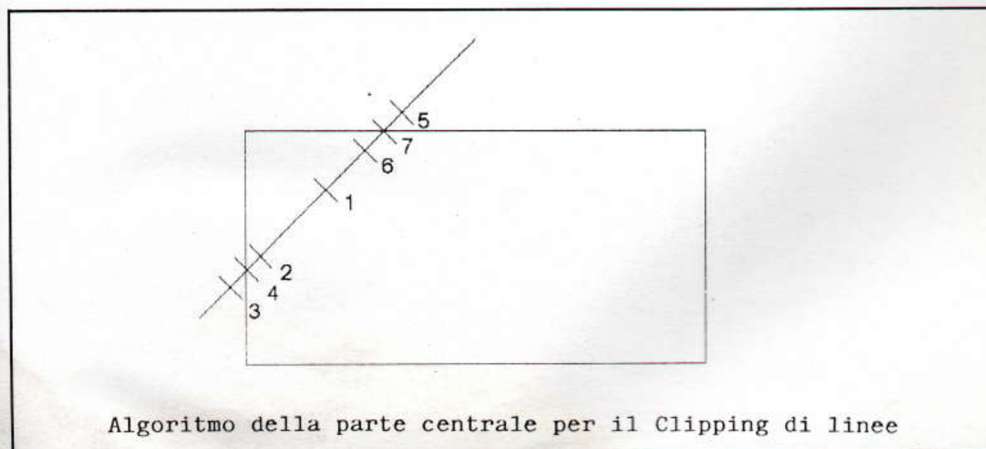


Fig. 3.21 Suddivisione ricorsiva di una linea

Osserviamo brevemente il numero di passaggi necessari alla determinazione del punto di intersezione. Il numero di passaggi dipende naturalmente dalla lunghezza della linea. Esso potrà venire determinato tramite la seguente formula:

$2^d = n \quad <=> \quad d = \log_2(n)$

dove:

d	=	numero delle suddivisioni
n	=	numero dei punti della linea
$\log_2(n)$	=	logaritmo di 2 di n

Esempio: se una linea possiede 256 punti, sarà possibile determinare i punti di taglio in un massimo di $\log_2(256) = 8$ passaggi. In caso di linee con 512 punti, sono necessari un massimo di 9 passaggi e per 1024 punti i passaggi necessari sono 10. Si tratta quindi di valori assolutamente accettabili.

CAPITOLO 4

Ingresso nel mondo Tridimensionale

Avendo posto le basi nei capitoli precedenti, possiamo finalmente addentrarci nello spazio. Disponiamo infatti delle premesse essenziali per poter comprendere il mondo tridimensionale: molto di ciò che seguirà non ci apparirà quindi completamente nuovo. Al contrario, troveremo numerosi riferimenti alla rappresentazione bidimensionale di immagini. In caso di difficoltà di comprensione sarà sufficiente tornare indietro di qualche pagina e andare a verificare come funzionava in due dimensioni.

Prima della lettura del capitolo è necessario avere preso dimestichezza con gli elementi di base bidimensionali, dal momento che da questo punto in poi se ne ipotizza una conoscenza approfondita.

4.1 Tutto è relativo: sistemi di coordinate

Al fine di descrivere il nostro mondo, cioè il nostro mondo immaginario che vogliamo rappresentare su monitor, stampante o Plotter, abbiamo bisogno di un sistema di coordinate spaziale. In questo sistema di coordinate ogni punto, ogni angolo ecc. ha una posizione esattamente specificata. Di conseguenza, ogni oggetto può venire indicato con la propria posizione e dimensione. Il suo colore o altre caratteristiche importanti per la sua rappresentazione vengono per il momento tralasciate.

In un sistema di coordinate bidimensionale, che abbiamo visto finora, cioè in grado di identificare solo tutti i punti su di un piano, utilizzavamo, per l'identificazione di un punto, solo due valori, cioè le cosidette coordinate. Come sappiamo tali valori rappresentano la componente x ed y della posizione, rispetto agli assi delle coordinate perpendicolari tra loro.

Nel sistema di coordinate tridimensionale si aggiunge un terzo asse, l'asse z, il quale è anch'esso perpendicolare agli altri due assi di coordinate, quindi è come se entrasse nel piano.

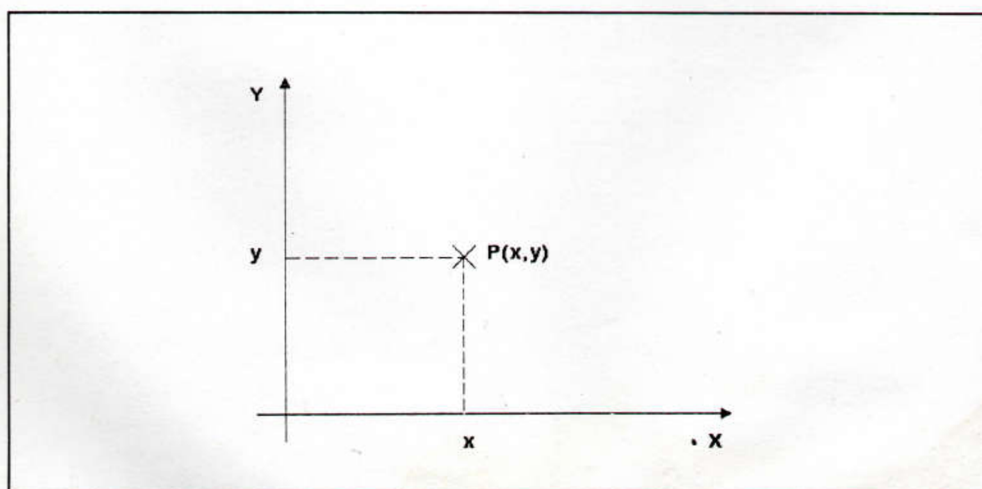


Fig. 4.1 Sistema di coordinate bidimensionali

Da questo punto di vista esistono due possibilità: che l'asse z fuoriesca dal piano (sistema destrorso) oppure penetri in esso (sistema sinistro).

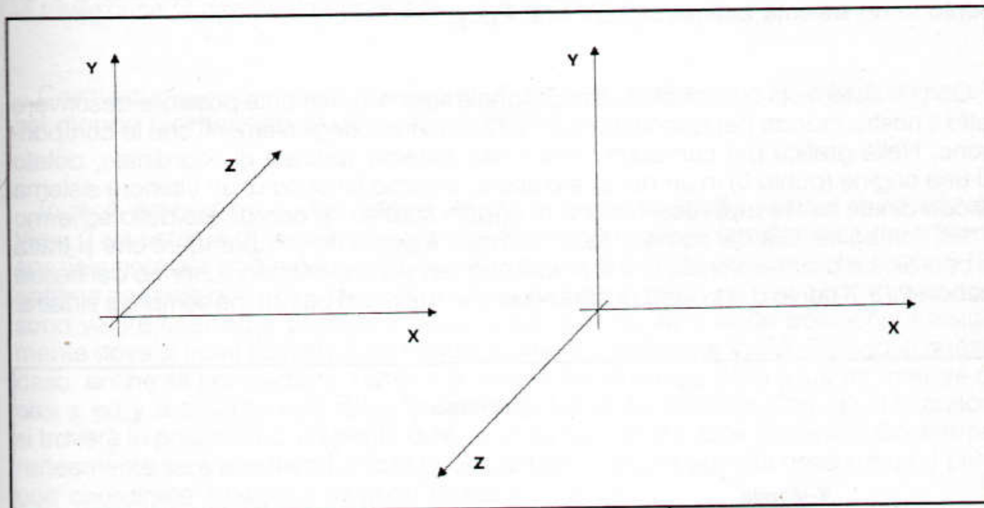


Fig. 4.2 Sistema destrorso e sistema sinistrorso

Quindi le coordinate z dei due sistemi di coordinate si differenziano l'una dall'altra solo nel segno che le precede. In qusto libro lavoreremo quasi esclusivamente con il sistema sinistrorso, cosa ormai usuale nella grafica per computer e anche in ampie parti della matematica. Non è affatto un problema trasferire tutti i calcoli, le matrici e le coordinate in un sistema destrorso (come abbiamo già visto, le coordinate x ed y non cambiano, solo la coordinata z cambierà di segno).

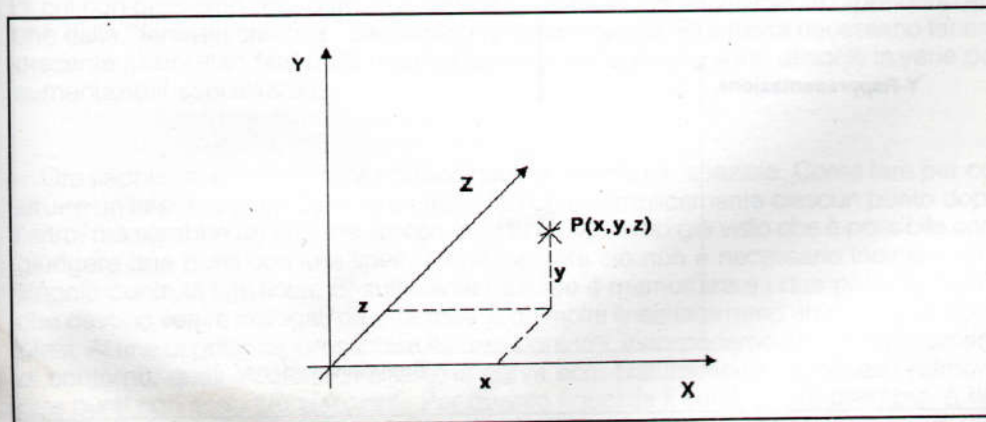


Fig. 4.3 Individuazione determinazione di un punto in un sistema di coordinate tridimensionale

Se vogliamo indicare un punto in un sistema di coordinate spaziale, abbiamo bisogno in totale di tre diversi valori: le coordinate x, y e z (tali coordinate forniscono la distanza del punto dall'origine delle coordinate in direzione x, y e z). Quindi indicheremo un punto tridimensionale come segue: P(x,y,z). Ricordiamo che l'indicazione di un punto in un sistema bidimensionale era: P(x,y).

Con un sistema di coordinate tridimensionale sarà naturalmente possibile descrivere tutto il nostro mondo per quanto concerne la posizione degli elementi che lo compongono. Nella grafica per computer, oltre a tale sistema spaziale di coordinate, dotato di una origine (punto 0) in un punto a piacere, avremo bisogno di un ulteriore sistema di coordinate per la rappresentazione di oggetti spaziali: le coordinate dello schermo o dell'immagine. Già dal nome di tali coordinate è possibile comprendere che si tratta di coordinate bidimensionali, che si riferiscono semplicemente allo schermo del nostro elaboratore. Il punto 0 si troverà di conseguenza in alto o in basso, ma sempre a sinistra.

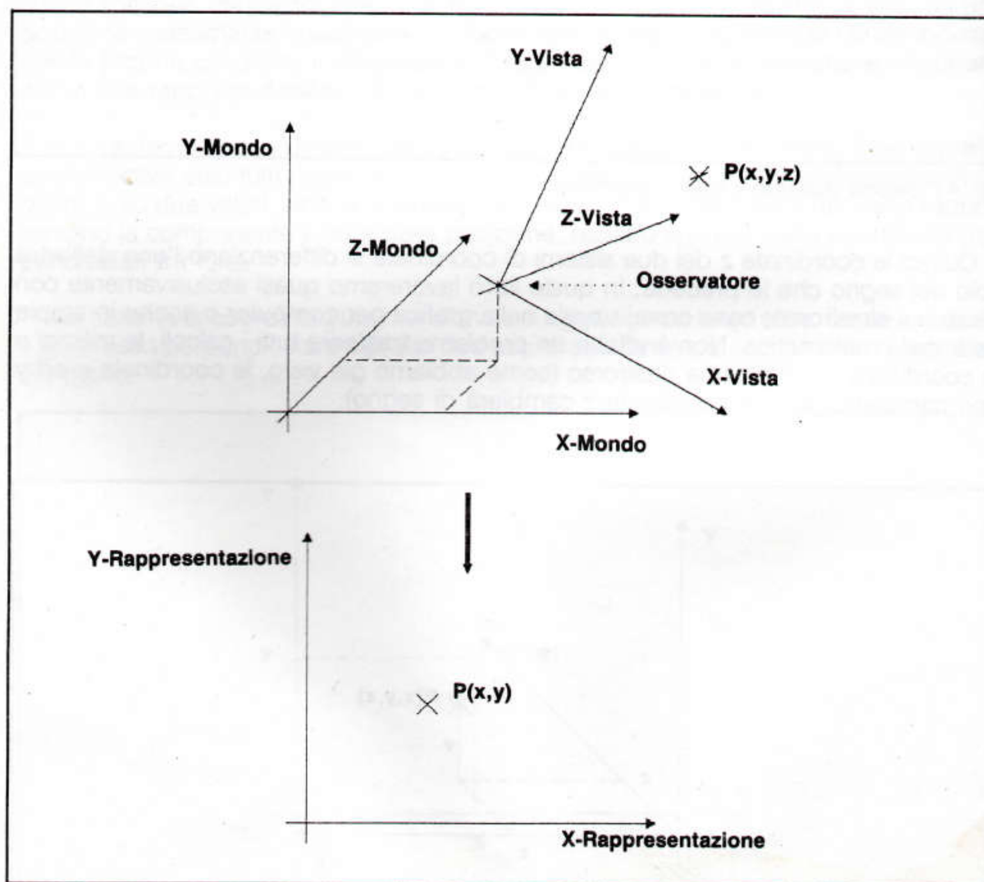


Fig. 4.4 Rapporti fra mondo, vista e rappresentazione

Le coordinate tridimensionali del sistema spaziale devono venire trasformate prima o poi nelle coordinate bidimensionali dello schermo. Le operazioni matematiche necessarie ad effettuare ciò verranno affrontate in questo capitolo più in seguito. Naturalmente anche in questo caso, un ruolo importante è rivestito dalle trasformazioni quali la traslazione, il cambiamento di scala e la rotazione.

Costruiamo quindi il nostro mondo in un sistema tridimensionale e trasformiamolo nel mondo bidimensionale del nostro schermo.

In molti programmi di computer si utilizza ancora il cosidetto sistema di coordinate View. Tale sistema di coordinate, anch'esso tridimensionale, si differenzia dal sistema spaziale normale solo nella posizione del punto 0. Mentre la posizione del punto 0 nel sistema spaziale può trovarsi in un punto qualunque, cioè gli assi delle coordinate possono venire orientati a piacere, il punto 0 del sistema View viene posizionato esattamente dove si trova l'osservatore in quel momento (posizione View). Spesso in questo caso, anche se non sempre, l'asse z si trova nella direzione dello sguardo, mentre gli assi x ed y si trovano verticalmente e orizzontalmente ad esso. Con ciò l'utilizzatore si troverà in possesso di un punto di riferimento ben chiaro, cioè se stesso. Contemporaneamente sarà altrettanto chiaro quali parti dello spazio egli stia osservando. I punti con coordinate z negativi saranno quindi invisibili.

4.2 Struttura dei dati di un mondo spaziale

Stiamo per addentrarci nella costruzione, per il momento ancora grossolana, di un mondo tridimensionale, come quello che viene realizzato dai normali programmi per CAD. Cerchiamo tuttavia di non spaventarci: per le nostre applicazioni "private" non sarà necessario utilizzare tale struttura completamente. La struttura che stiamo per descrivere è quasi del tipo ideale. Esistono naturalmente moltissime alternative e varianti, di cui non possiamo occuparci in questo momento, e che parzialmente dipendono anche dalla "fantasia creativa" del singolo programmatore. E' tuttavia necessario tenere presente alcuni standard della memorizzazione dei dati, che sono descritti in varie documentazioni specializzate.

Ora sappiamo come definire i singoli punti di un mondo spaziale. Come fare per costruire un'intera scena? Certo potremmo indicare semplicemente ciascun punto dopo l'altro, ma sarebbe un enorme spreco di tempo. Abbiamo già visto che è possibile congiungere due punti con una linea, e che per fare ciò non è necessario indicare ogni singolo punto di tale linea. E' sufficiente indicare e memorizzare i due punti terminali, che devono venire collegati da una linea. Con molte linee otterremo un'immagine completa. Al fine di poter rappresentare anche i contorni, incorporiamo anche altri elementi di contorno, quali il cerchio, l'ellisse, la curva ecc. Naturalmente per questi elementi due punti non sono più sufficienti. Per quanto riguarda il cerchio, per esempio, è necessario indicare il centro ed un raggio, per l'ellisse sono necessari il centro e due raggi, per gli archi saranno necessari gli angoli di partenza e di fine ecc.

Nei sistemi spaziali tutto ciò non è più sufficiente. I programmi tridimensionali riducono molti di tali elementi di contorno a superfici intere. Una superfice può quindi venire limitata da più linee o combinazioni di linee e archi ecc.

Quindi, esattamente come un punto può far parte di più linee, un elemento di contorno può venire utilizzato da più superfici. Molti problemi tridimensionali (quali per esempio l'eliminazione di linee e superfici nascoste, oppure l'ombreggiatura di superfici) ipotizzano la disponibilità di tali definizioni di superfici.

Più superfici insieme formano quindi un cosidetto oggetto (parallelepipedo, sfera, cono, piramide ecc). Più oggetti possono venire in seguito raggruppati insieme, fino a determinare oggetti più complessi. Con ciò otteniamo un sistema di dati piuttosto complicato, ma particolarmente efficiente, che dovremo tenere presente al momento dei calcoli spaziali. Questo sistema orientato agli oggetti ha notevoli vantaggi: infatti è sufficiente sviluppare tali oggetti una volta per tutte. Quindi sarà possibile memorizzarli in una biblioteca di oggetti su disco o disco rigido. Tali oggetti pronti potranno in seguito venire incorporati dall'utente nell'immagine che sta creando, ingranditi, ridotti, spostati, ruotati ecc. Una volta incorporato, un oggetto di questo genere potrà comunque venire modificato o addirittura eliminato. Con ciò si garantisce la massima flessibilità.

I sistemi CAD servono per costruire delle immagini composte dai più diversi oggetti di base, quali parallelepipedi, cubi, coni ecc, raggruppati insieme per formare immagini più complesse. Inoltre gli utilizzatori sono anche in grado di dichiarare tali immagini come nuovi oggetti (es. una casa) ed utilizzarli a loro volta per la costruzione di oggetti ancora più complessi (es. un'intera città). L'utente di un programma CAD vede quindi solo gli oggetti, e non ha più bisogno di occuparsi degli elementi di un oggetto (superfici, linee, punti).

Prendiamo come esempio un piccolo nucleo abitato: tale nucleo abitato è composto da oggetti chiaramente identici: le case. Ogni singola casa, come quelle dei giochi per bambini, è composta da diversi oggetti di base (tetto, un parallelepipedo come base ecc) o da altri oggetti composti. Ciascuno di questi oggetti di base è composto a sua volta dalle superfici che lo delimitano, le quali a loro volta vengono definite dai loro lati. Nel livello ancora più basso troveremo gli estremi dei lati oppure gli altri parametri determinanti, quali per esempio il raggio in caso di cerchi ecc.

Dal momento che l'utilizzatore sta lavorando con oggetti liberamente spostabili, copiabili e duplicabili, sarà opportuno allestire un sistema di coordinate proprio per ogni oggetto. A questo scopo è necessario fornire ancora una coordinata, se vogliamo determinare la posizione di questo oggetto nel sistema spaziale, oppure, in caso di sub-oggetti, nel sistema di oggetti precedente, cioè le coordinate del punto 0 del sistema di oggetti. A ciò si aggiungono ancora gli angoli degli assi dei sistemi di oggetti rispetto agli assi dei sistemi spaziali (cioè agli assi dei sistemi spaziali dell'oggetto precedente).

All'interno di questo sistema di coordinate dell'oggetto sono contenuti solo i diversi sub-oggetti ed elementi (superfici, punti ecc) di cui è composto l'oggetto (tutto ciò naturalmente rispetto al sistema di oggetti). E' solo al momento in cui questo nostro mondo spaziale deve venire tracciato, che le coordinate interne all'oggetto vengono trasformate nelle coordinate del sistema di oggetti successivo e quindi in coordinate spaziali.

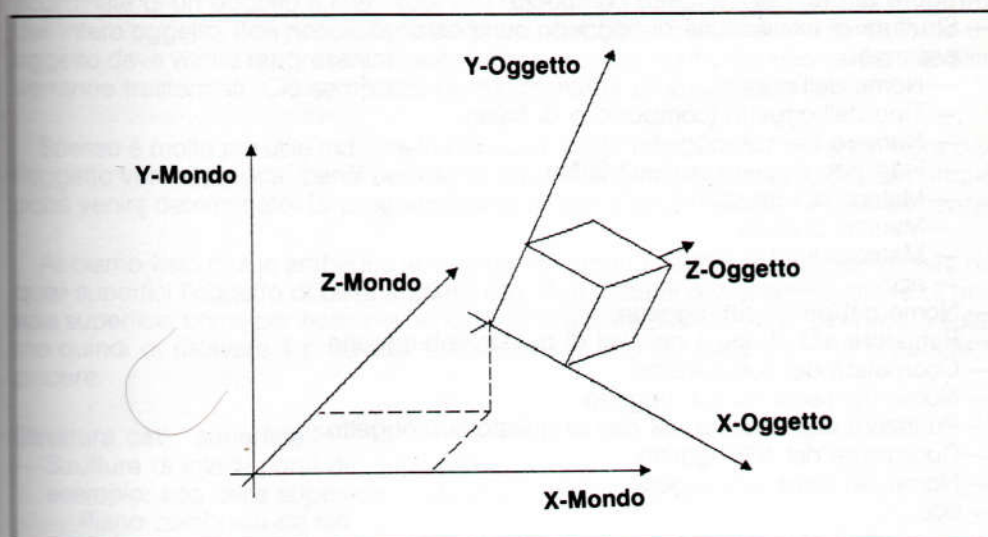


Fig. 4.5 Un oggetto [in un sistema di "mondo"] nello spazio con riferimenti relativi ed assoluti

In tal modo un oggetto resta sempre e comunque accessibile nello spazio (spostabile, cancellabile ecc).

I dati di uno "spazio" tipico saranno circa i seguenti:

Dapprima esiste un blocco di dati principali chiamato "spazio". In esso sono contenuti semplicemente i nomi (o numeri) degli oggetti principali, di cui tale spazio è composto (quest'ultimo può essere una sola unità, a sua volta composta da diversi oggetti). A ciò si aggiungono le coordinate di tali oggetti nel sistema spaziale. Da ogni immissione deve essere deducibile, dove è possibile trovare i dati che descrivono più precisamente il singolo oggetto (es. un puntatore ad una struttura di dati oggetto):

Struttura di dati per "spazio":
— Struttura di intestazione di "spazio" (es. nome dello spazio, numero degli oggetti, ecc.)
— Nome del primo oggetto
— Puntatore alla struttura di dati di tale oggetto
— Coordinate dell'oggetto
— Nome del secondo oggetto
— Puntatore alla struttura di dati di tale oggetto
— Coordinate dell'oggetto
— Nome del terzo oggetto
— ecc.

Ogni oggetto composto verrà descritto tramite un blocco di dati simile:

Struttura di dati per "oggetto composto":

— Struttura di intestazione di "oggetto composto"
  esempio:
  — Nome dell'oggetto
  — Tipo dell'oggetto (composto o di base)
  — Numero dei sub-oggetti
  — Flag per: oggetto visibile? Si/No
  — Matrice di traslazione
  — Matrice di scala
  — Matrice di rotazione
  — ecc.
— Nome del primo sub-oggetto
— Puntatore alla struttura dei dati di questo sub-oggetto
— Coordinate del sub-oggetto
— Nome del secondo sub-oggetto
— Puntatore alla struttura dei dati di questo sub-oggetto
— Coordinate del sub-oggetto
— Nome del terzo sub-oggetto
— ecc.

Gli oggetti di base, al contrario, non saranno più composti da sub-oggetti. Essi saranno composti invece da singoli superfici:

Struttura di dati per "oggetto di base":

— Struttura di intestazione di "oggetto di base"
  esempio:
  — Nome dell'oggetto
  — Tipo dell'oggetto (composto o oggetto di base)
  — Flag per: oggetto visibile? Si/No
  — Matrice di traslazione
  — Matrice di scala
  — Matrice di rotazione
  — ecc.
— Puntatore alla struttura di definizione della prima superficie
— Puntatore alla struttura di definizione della seconda superficie
— ecc.

La struttura di intestazione (Header) di un oggetto contiene informazioni che possono essere molto importanti. Dal momento che le coordinate delle diverse parti di un oggetto (sub-oggetto, punto, linea, superficie, ecc) dovrebbero essere fisse e non modificabili (ad eccezione delle modifiche dirette agli oggetti), sarà necessario indicare per quali valori l'intero oggetto, compreso il sistema di coordinate successivo oppure l'oggetto successivo (esempio: sistema spaziale) dovrà venire spostato, ingrandito o ruotato. Ciò accade tramite le matrici di trasformazione che abbiamo già visto.

Questo sistema ha diversi vantaggi. Il primo è che tutti gli oggetti restano fissi e necessitano di venire definiti una volta sola, anche se sono presenti più volte nella stessa immagine. Ciò fa risparmiare spazio in memoria. Un ulteriore vantaggio è che tutte le

coordinate di un oggetto e tutti i suoi sub-oggetti, al momento di una trasformazione dell'intero oggetto, non necessitano di venire ricalcolate. E' solo al momento in cui un oggetto deve venire rappresentato sul monitor che i punti, che devono venire tracciati, verranno trasformati. Ciò semplifica notevolmente la programmazione.

Spesso è molto più utile indicare le matrici di trasformazione non nella struttura dell'oggetto vera e propria, bensì nella struttura dell'oggetto successivo. Ciò comunque potrà venire determinato dal programmatore stesso a seconda delle proprie esigenze.

Abbiamo visto che in ambedue le strutture di oggetto sarà possibile determinare da quali superfici l'oggetto di base è composto. Può trattarsi per esempio anche di una sola superfice, come per esempio nel caso di una sfera o di un piano semplice. Vediamo quindi di risolvere il problema della struttura di definizione per una superfice a piacere:

Struttura dati "superfice":
— Struttura di intestazione di "superfice"
  esempio: tipo della superfice:
  — Piano composto da lati
  — Sfera, ellissoide, per i quali devono venire indicati solo il centro ed i raggi
  — Superfice curva, che deve venire determinata da una funzione o da diversi punti di appoggio
— Ulteriori caratteristiche determinanti a seconda del tipo di superfice
  esempio:
  — Puntatore alla struttura di definizione del primo lato
  — Puntatore alla struttura di definizione del secondo lato
  — ecc.
  oppure:
  — Coordinate del centro
  — Raggio della sfera

Vediamo quindi che già in questa fase viene effettuata una differenziazione a seconda del tipo di superfice. I tipi di superfici principalmente utilizzati sono le superfici piane, rettangolari, composte da quattro lati. Sono tuttavia possibili numerosissimi altri tipi di superfice (es: sfera, semisfera, ecc). Anche questi tipi devono venire determinati nella struttura della superfice. Di solito tuttavia ci si limita, come abbiamo già visto, alle superfici piane facili da manipolare (tuttavia anche le sfere possono venire approssimate da questo tipo di superfici).

Tali superfici piane sono composte quindi da diversi lati (almeno tre), per i quali devono venire indicati a loro volta due estremi ciascuno:

Struttura dati "lati":
— Numero del primo estremo
— Numero del secondo estremo

Sarà possibile trovare le coordinate solo dopo aver numerato completamente gli estremi (naturalmente relativamente al sistema di coordinate dell'oggetto in vigore in quel momento).

Nelle strutture precedenti abbiamo citato i "puntatori", i quali naturalmente possono anche essere numeri o simili (come nel caso dei lati). Inoltre, se serve, è possibile conglobare le diverse strutture di dati. Le strutture più elevate (oggetti, superfici ecc.) devono venire tralasciate in caso, per esempio, di un cosidetto modello di tracciatura di un oggetto. Inoltre sarà necessario fare attenzione al fatto che il sistema di strutture o di oggetti non divenga troppo complicato, dal momento che tale eventuale complicazione, in presenza di una organizzazione non ancora perfetta, potrebbe far aumentare sostanzialmente i tempi di calcolo.

Ritroveremo spesso nel presente libro alcune parti del sistema di dati di cui sopra, in particolare laddove sono necessarie per una migliore comprensione.

4.3 Elementi matematici di base per il calcolo tridimensionale

Il calcolo matriciale e la sua applicazione nella grafica per computer dovrebbe già essere noto, grazie al capitolo relativo alle due dimensioni. Tuttavia, per le molte applicazioni grafiche non possiamo ignorare una ulteriore tecnica: il calcolo vettoriale.

Esistono in realtà diversi metodi matematici per la produzione delle più diverse figure ed effetti grafici. Ritengo tuttavia che il calcolo vettoriale sia la forma di rappresentazione più semplice e comprensibile, come vedremo fra poco. Chiedo scusa ai lettori ferrati in questa materia per il linguaggio poco matematico che sto per usare, che renderà tuttavia molto più comprensibile questo argomento ai lettori più incerti.

a) Cos'è un vettore?

Cerchiamo di immaginare una freccia che punta in una determinata direzione e che possiede una determinata lunghezza. Al fine di identificare un vettore, dobbiamo fornire due parametri: la direzione e la lunghezza. Questa freccia potrà trovarsi

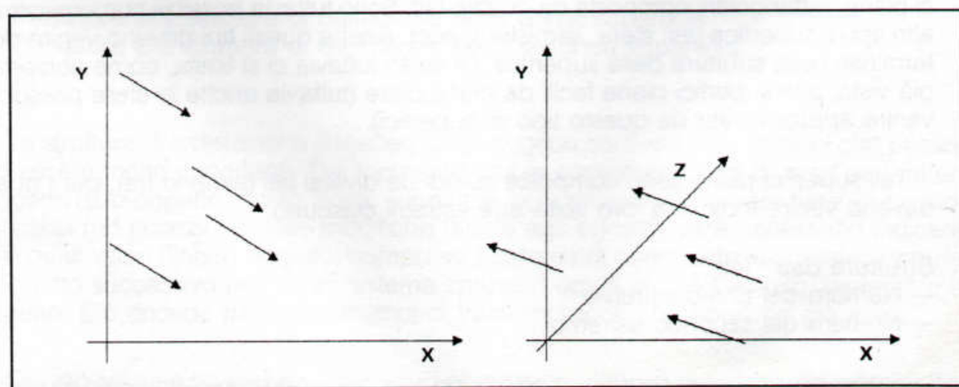


Fig. 4.6 Vettore translato nel piano e nello spazio

in una posizione qualunque sul piano (o nello spazio) senza che sia necessario modificare questi due parametri. Quindi è possibile spostare tale freccia a piacere su di un piano o nello spazio. Esso resterà sempre lo stesso vettore, se non si modifica la lunghezza e la direzione. Il punto iniziale di un vettore viene chiamato anche piede, mentre il punto terminale viene chiamato punta.

In futuro contrassegneremo sempre i vettori con una piccola freccia diretta verso destra (es: $\vec{v}$). I numeri semplici, al contrario, verranno espressi in maniera normale.

Nel caso in cui un vettore abbia una lunghezza Uno, lo si chiamerà vettore unitario. Il vettore zero ha lunghezza Zero, la sua direzione è indeterminata.

Con i vettori è possibile anche effettuare dei calcoli: la direzione di un vettore cambia per esempio nella direzione contraria, quando se ne modifica il segno ($-\vec{v}$). $\vec{v}$ e $-\vec{v}$ sono detti anche antiparalleli (infatti essi sono paralleli ma puntano in direzioni contrarie):

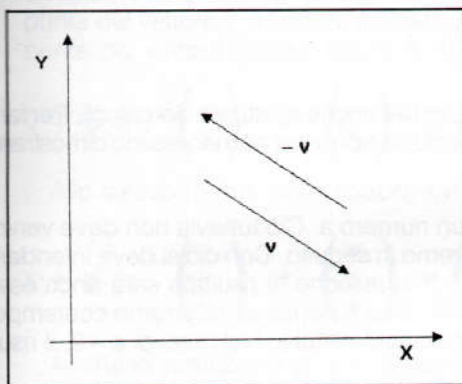


Fig. 4.7 Vettori $\vec{v}$ e $-\vec{v}$

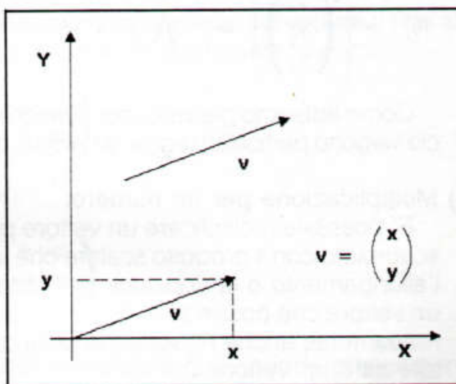


Fig. 4.8 Rappresentazione di un vettore nel piano cartesiano

In alternativa è possibile indicare un vettore $\vec{v}$ anche con coordinate semplici. Per fare ciò, si ipotizza che il piede del vettore si trovi nel punto 0 delle coordinate (è possibile tuttavia spostarlo a piacere). Come coordinate del vettore verranno quindi indicate le coordinate della sua punta (anche in questo caso sono sufficienti due parametri per il piano e tre per lo spazio):

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

oppure, nello spazio:

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Questo sistema di scrittura delle coordinate è molto importante, dal momento che lo utilizzeremo nei programmi. Determineremo inoltre delle parallele alle matrici del punto. Un vettore infatti può anche venire visto come una specie di matrice. Ciò nonostante sarà importante non scambiare questi due concetti matematici, dal momento che vengono utilizzati in maniera completamente diversa l'uno dall'altro.

La lunghezza del vettore è la sua grandezza (una sorta di valore assoluto per vettori). Essa può venire determinata tramite la seguente formula, che dovremo tenerci bene a mente, dal momento che ne avremo bisogno spesso:

$$|\vec{v}| = \left| \begin{pmatrix} v_x \\ v_y \end{pmatrix} \right| = \sqrt{(v_x x^2 + v_y^2)}$$

Rileveremo giustamente delle somiglianze con il teorema di Pitagora. Tale formula può venire dedotta dallo schizzo precedente.

In caso di vettori spaziali avremo invece:

$$|\vec{v}| = \left| \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \right| = \sqrt{(v_x^2 + v_y^2 + v_z^2)}$$

Come abbiamo già visto, con i vettori è possibile anche effettuare dei calcoli. Per fare ciò valgono particolari regole di calcolo, che tuttavia non è per noi necessario dimostrare:

b) Moltiplicazione per un numero:

E' possibile moltiplicare un vettore per un numero a. Ciò tuttavia non deve venire scambiato con il prodotto scalare che vedremo in seguito. Con ciò si deve intendere l'allungamento o la riduzione del vettore $\vec{v}$ in questione. Il risultato sarà anch'esso un vettore che possiederà la lunghezza $a^*|\vec{v}|$. Se a è negativo, otterremo contemporaneamente anche l'inversione della direzione del vettore. Nel caso di a = 0, il risultato sarà un vettore 0:

$$a^*\vec{v} = a^* \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} a^*v_x \\ a^*v_y \end{pmatrix}$$

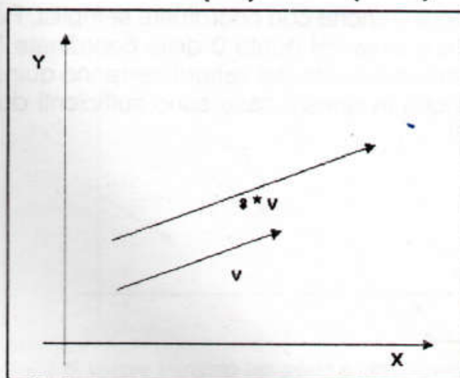


Fig. 4.9 Moltiplicazione per un numero

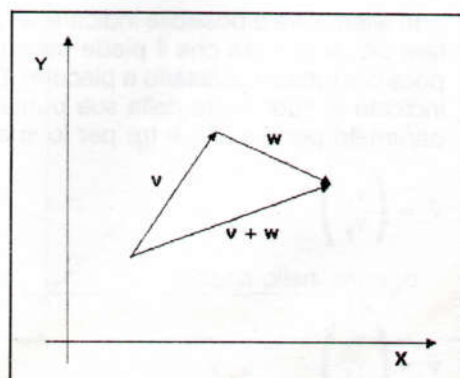


Fig. 4.10 Addizione vettoriale

Nello spazio:

$$a^*\vec{v} = a * \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} a^*v_x \\ a^*v_y \\ a^*v_z \end{pmatrix}$$

In questo contesto è molto importante un ulteriore concetto: l'indipendenza lineare dei vettori. Due vettori saranno indipendenti linearmente quando non è possibile trasformare uno nell'altro tramite una moltiplicazione per un numero semplice. In altre parole quando non sono nè paralleli nè antiparalleli l'uno rispetto all'altro. Di conseguenza verranno definiti dipendenti linearmente, o meglio collineari, due vettori che hanno la stessa direzione (o direzioni opposte) ma non necessariamente la stessa lunghezza.

c) Addizioni/sottrazioni di vettori:

Con due vettori $\vec{v}$ e $\vec{w}$ è possibile anche effettuare delle addizioni e delle sottrazioni. Il risultato sarà anch'esso un vettore. Se vogliamo rappresentare graficamente un'addizione di questo genere, dovremo spostare solo il piede del vettore $\vec{w}$ sulla punta del vettore $\vec{v}$. Il vettore risultante partirà quindi dal piede del vettore $\vec{v}$ fino alla punta del vettore $\vec{w}$ (vedi figura 4.10). Sotto forma di coordinate, avremmo:

$$\vec{v} + \vec{w} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} + \begin{pmatrix} w_x \\ w_y \end{pmatrix} = \begin{pmatrix} v_x + w_x \\ v_y + w_y \end{pmatrix}$$

Allo stesso modo, nello spazio, avremo:

$$\vec{v} + \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} + \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_x + w_x \\ v_yy + w_y \\ v_z + w_z \end{pmatrix}$$

Anche la sottrazione di due vettori è molto semplice e corrisponde praticamente all'addizione. Al momento del tracciato invertiremo semplicemente il segno del secondo vettore ($-\vec{w}$ => cambio di direzione) e effettueremo l'addizione dei due vettori ($\vec{v} + (-\vec{w})$).

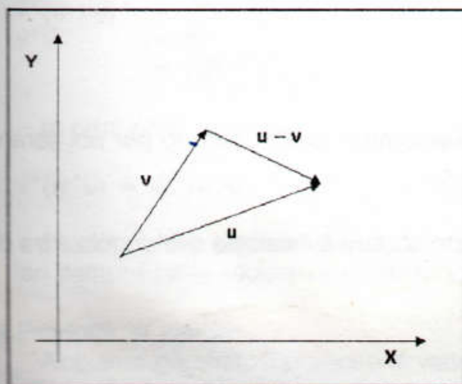


Fig. 4.11 Sottrazione vettoriale

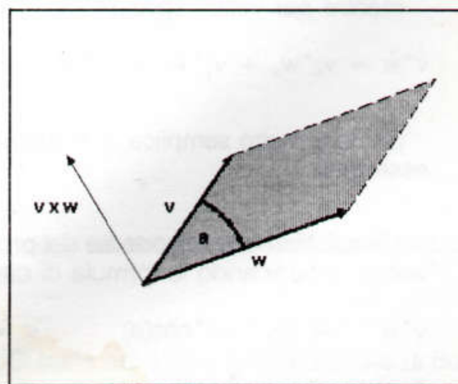


Fig. 4.12 Prodotto vettoriale

$$\vec{v} + \vec{w} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} - \begin{pmatrix} w_x \\ w_y \end{pmatrix} = \begin{pmatrix} v_x - w_x \\ v_y - w_y \end{pmatrix}$$

Nello spazio:

$$\vec{v} + \vec{w} = \begin{pmatrix} v_x \\ v_yy \\ v_z \end{pmatrix} - \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_x - w_x \\ v_y - w_y \\ v_z - w_z \end{pmatrix}$$

Regole di calcolo:

$\vec{v}+\vec{w}$	$= \vec{w}+\vec{v}$	$\vec{v}+(\vec{w}+\vec{u})$	$= (\vec{v}+\vec{w})+\vec{u}$
$a^*(b^*\vec{v})$	$= (a^*b)^*\vec{v}$	$(a+b)^*\vec{v}$	$= a^*\vec{v} + b^*\vec{v}$
$a^*(\vec{v}+\vec{w})$	$= a^*\vec{v} + a^*\vec{w}$		
$\lvert a^*\vec{v}\rvert$	$= \lvert a\rvert ^* \lvert\vec{v}\rvert$		

d) Prodotto scalare:

E' inoltre possibile moltiplicare l'uno per l'altro due vettori $\vec{v}$ e $\vec{w}$. In questo caso, tuttavia, abbiamo due possibilità. Per il cosidetto prodotto scalare $\vec{v}^*\vec{w}$ (cioè: $\vec{v}$ per $\vec{w}$) il risultato sarà un numero semplice (scalare), mentre per il prodotto di vettori $\vec{v}$ x $\vec{w}$ (cioè: $\vec{v}$ attraverso $\vec{w}$) avremo di nuovo un vettore.

Il prodotto scalare viene definito come segue:

$$\vec{v}^*\vec{w} = |\vec{v}| ^* |\vec{w}| ^* \cos(a)$$

In questo caso, a è l'angolo tra i due vettori da moltiplicare. Le grandezze di $\vec{v}$ e $\vec{w}$ sono, come noto, le lunghezze dei due vettori. In formato coordinate avremo:

$$\vec{v}^*\vec{w} = v_x{}^*w_x + v_y{}^*w_y$$

mentre per vettori spaziali:

$$\vec{v}^*\vec{w} = v_x{}^*w_x + v_y{}^*w_y + v_z{}^*w_z$$

Sembra molto semplice, e si tratta delle formule che in seguito per noi saranno essenziali.

Un'applicazione importante del prodotto scalare è il calcolo dell'angolo a tra due vettori, modificando la formula di cui sopra:

$$\vec{v}^*\vec{w} = |\vec{v}| ^* |\vec{w}|^*\cos(a) \quad <=>$$

$$\cos(a) = \frac{\vec{v} ^* \vec{w}}{|\vec{v}| ^* |\vec{w}|} \quad <=>$$

$$\cos(a) = \frac{v_x * w_x + v_y * w_y + v_z * w_z}{\sqrt{((v_x^2 + v_y^2 + v_z^2) * (w_x^2 + w_y^2 + w_z^2))}}$$

Per vettori su di un piano, impostiamo semplicemente v_z e $w_z = 0$.

Per le nostre esigenze è molto importante anche un'altra caratteristica del prodotto scalare. Non appena i due vettori da moltiplicare si trovano l'uno ortogonale all'altro, il coseno del loro angolo (coseno di 90°) diventa 0 e di conseguenza il prodotto totale sarà:

$\vec{v} * \vec{w} = 0$ per $\vec{v}$ si intende verticalmente a $\vec{w}$

Quindi è possibile determinare con facilità se due vettori sono perpendicolari.

Esistono naturalmente numerosi vettori perpendicolari rispetto ad un altro vettore. Su di un piano, però, ce ne sono solo due, che possiedano contemporaneamente anche la stessa lunghezza (ciò può venire calcolato dal formato di coordinate del prodotto scalare):

vettore di partenza: $\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$

Rispetto ad esso avremo verticalmente e con la stessa lunghezza:

$\vec{v}_1 = \begin{pmatrix} v_y y \\ -v_x \end{pmatrix}$ e $\vec{v}_2 = \begin{pmatrix} -v_y \\ v_x \end{pmatrix}$

Per lo spazio tuttavia esistono moltissime possibilità in più.

Regole di calcolo:

$$\begin{aligned} \vec{v} * \vec{w} &= \vec{w} * \vec{v} \\ (a * \vec{v}) * \vec{w} &= a * (\vec{v} * \vec{w}) \\ \vec{v} * (\vec{w} + \vec{u}) &= \vec{v} * \vec{w} + \vec{v} * \vec{u} \\ \vec{v} * \vec{v} &= \vec{v}^2 \\ &= |\vec{v}|^2 \end{aligned}$$

In ogni caso non vale:

$\vec{v} * (\vec{w} * \vec{u}) = (\vec{v} * \vec{w}) * \vec{u}$

In altre parole: sarà necessario fare attenzione ad eseguire le moltiplicazioni scalari sempre nella sequenza giusta.

e) Prodotto di vettori:

Abbiamo già citato il prodotto di vettori. Si tratta del tipo di moltiplicazione di due vettori che fornisce come risultato un altro vettore. Questo nuovo vettore $\vec{p} = \vec{v} \times \vec{w}$ ha una lunghezza di:

$$|\vec{p}| = |\vec{v} \times \vec{w}| = |\vec{v}| * |\vec{w}| * \sin(a)$$

Ciò non è altro che il contenuto in superfice del parallelogramma tracciato da $\vec{v}$ e $\vec{w}$ (vedi figura 4.12). $\vec{p}$ è perpendicolare ai due vettori $\vec{v}$ e $\vec{w}$, quindi è proiettato nello spazio ($\vec{v}$, $\vec{w}$ e p producono un sistema destro). Un vettore perpendicolare viene chiamato anche vettore normale.

Matematicamente viene definito come segue:

$$\vec{p} = \vec{v} \times \vec{w} = \begin{pmatrix} v_y * w_z - v_z * w_y \\ v_z * w_x - v_x * w_z \\ v_x * w_y - v_y * w_x \end{pmatrix}$$

Regole di calcolo:

$\vec{v} \times \vec{w} = -(\vec{w} \times \vec{v})$
$(a * \vec{v}) \times \vec{w} = a * (\vec{v} \times \vec{w})$
$\vec{v} \times (\vec{w} + \vec{u}) = \vec{v} \times \vec{w} + \vec{v} \times \vec{u}$
$\vec{v} \times \vec{w}$ = Vettore zero (nel caso in cui $\vec{v}$ sia parallelo o antiparallelo a $\vec{w}$)
$\vec{v} \times \vec{v}$ = Vettore zero

f) Rappresentazione dei punti tramite vettori:

Il sistema dei vettori è molto flessibile. Esso ci fornisce per esempio la modalità per rappresentare un singolo punto in un sistema di coordinate, anche in notazione vettoriale, senza problemi (in questo caso sono presenti sia matrici che vettori).

Il trucco è molto semplice: si pensi dapprima ad un punto P con le coordinate P(x,y). Inoltre si immagini un vettore $\vec{v}$ dall'origine dello coordinate fino al punto P desiderato in un sistema di coordinate. Tale vettore potrà venire indicato, come noto, con la notazione di coordinate nel modo seguente:

$$\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{oppure:} \quad \vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

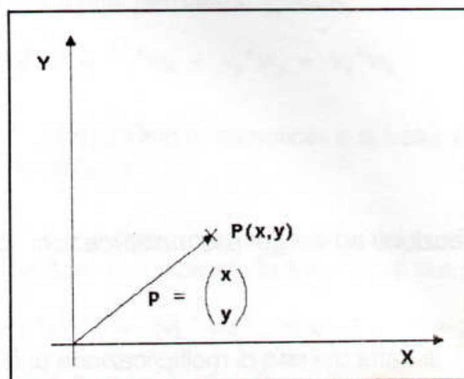


Fig. 4.13 Rappresentazione di un punto tramite vettori

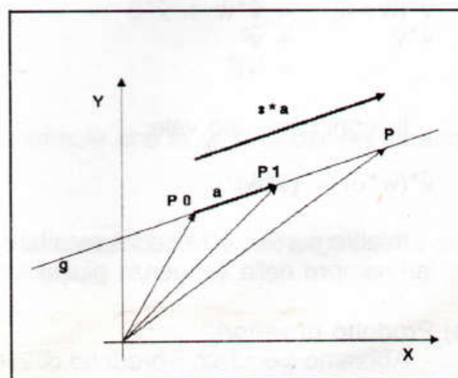


Fig. 4.14 Equazione vettoriale di una retta

x, y e z sono quindi le coordinate del punto. Invece di effettuare i calcoli con il punto stesso, facciamolo con il vettore di questo punto, senza incontrare alcun problema. Potremo anche cambiare il nome del vettore $\vec{v}$ in P:

$$P = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{oppure:} \quad P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Con P (che rappresenta propriamente il punto P) effettueremo i calcoli esattamente come con vettori normali - e vedrete che lo utilizzeremo molto.

g) Rappresentazione di linee tramite vettori:

Per i calcoli geometrici che ci accingiamo a compiere, le linee sono naturalmente molto importanti, in particolare le rette. Per questo motivo vediamo subito come può venire rappresentata sotto forma di vettori una retta bidimensionale. Osserviamo, per migliore chiarezza, la figura 4.14.

Equazione vettoriale di una retta (vale sia per il piano che per lo spazio):

$$g: \vec{p} = \vec{p}_0 + t^* \vec{a}$$

oppure, in notazione di coordinate:

$$g: \begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} p_{0x} \\ p_{0y} \end{pmatrix} + t^* \begin{pmatrix} a_x \\ a_y \end{pmatrix}$$

$$= \begin{pmatrix} p_{0x} + t^* a_x \\ p_{0y} + t^* a_y \end{pmatrix}$$

Da ciò potremo derivare semplicemente anche due equazioni, calcolando separatamente le componenti x ed y:

$$g: \begin{aligned} p_x &= p_{0x} + t^* a_x \\ p_y &= p_{0y} + t^* a_y \end{aligned}$$

Nello spazio avremo inoltre la coordinata z:

$$g: \begin{aligned} p_x &= p_{0x} + t^* a_x \\ p_y &= p_{0y} + t^* a_y \\ p_z &= p_{0z} + t^* a_z \end{aligned}$$

Dove:

- $\vec{p}$ — Vettore dall'origine delle coordinate fino al punto P da calcolarsi della retta (oppure semplicemente: punto P)
- $\vec{p}_0$ — Vettore dall'origine delle coordinate ad un punto P_0 a piacere della retta (oppure semplicemente: punto P_0)
- $\vec{a}$ — "Vettore di direzione", vettore a piacere parallelo alla retta
- t — Fattori di "allungamento" (oppure riduzione) del vettore $\vec{a}$, al fine di raggiungere il punto P.

"g" significa che abbiamo a che vedere con una equazione di rette. Il punto (o meglio il vettore) $\vec{p}$ rappresenta quindi tutti i punti della linea.

Partiamo quindi da un punto P_0 sulla retta (a questo scopo può venire utilizzato un punto a piacere della retta). Dal momento che d'ora in poi esprimeremo i punti solo come vettori che vanno dall'origine delle coordinate fino al punto in questione (vedi sopra), da questo punto risulta il vettore $\vec{p}_0$ (per semplicità, anche se in maniera poco matematica, abbiamo dato al punto e al vettore lo stesso nome, dal momento che in seguito non vorremo più effettuare distinzioni tra il punto e il vettore). Tracciamo da questo punto un secondo vettore $\vec{a}$ in direzione della retta (anche questo vettore è stato scelto a piacere e deve solo giacere sulla retta, indipendentemente dalla sua lunghezza). Questo vettore ci fornisce la direzione della retta. Ora, al fine di calcolare un punto a piacere P sulla retta, dovremo moltiplicare questo vettore di direzione con un fattore t. Ambedue i vettori risultanti $\vec{p}_0$ e $t^*\vec{a}$ vengono sommati e forniscono come risultato il vettore $\vec{p}$, la cui forma parametrica ci fornisce direttamente le coordinate del punto della linea P.

Al fine di determinare una retta ben precisa, sarà quindi necessario fornire semplicemente i parametri $\vec{p}_0$ e $\vec{a}$. Da valori a piacere per t verranno calcolati quindi tutti i punti P (oppure vettori p) della retta.

h) Intersezione di due rette:

Con quanto appreso finora, potremo calcolare molto semplicemente il punto di intersezione di due rette su un piano. Ipotizziamo di avere due rette g e h con le seguenti equazioni:

$$g: \vec{p} = \vec{p}0 + t^*\vec{a}$$
$$h: \vec{q} = \vec{q}0 + s^*\vec{b}$$

Dal momento che cerchiamo un singolo punto, che si trovi su ambedue le rette, quindi per il quale p e q divengano uguali, potremo uguagliare le due equazioni:

$$\vec{p}_0 + t^*\vec{a} * \vec{q}_0 + s^*\vec{b}$$

In notazione di coordinate (su di un piano potremo impostare il parametro z=0 e quindi tralasciarlo) avremo:

$$\begin{pmatrix} p_{0x} \\ p_{0y} \\ p_{0z} \end{pmatrix} + t^* \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} q_{0x} \\ q_{0y} \\ q_{0z} \end{pmatrix} + s^* \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$$

oppure:

$p_{0x} + t^*a_x = q_{0x} + s^*b_x$ e
$p_{0y} + t^*a_y = q_{0y} + s^*b_y$ e
$p_{0z} + t^*a_z = q_{0z} + s^*b_z$

Dal momento che conosciamo i parametri p_{0x}, p_{0y}, p_{0z}, a_x, a_y, a_z, nonché q_x, q_y, b_x, b_y e b_z (essi infatti determinano le due rette) dovremo determinare ancora t e s, al fine di ottenere il punto di intersezione P o Q. Risolviamo due delle equazioni secondo una delle due variabili (in questo caso secondo t):

$$t = \frac{q_{0x} + s^*b_x - p_{0x}}{a_x}$$

e:

$$t = \frac{q_{0y} + s^*b_y - p_{0y}}{a_y}$$

Uguagliamo di nuovo le due equazioni, al fine di poter eliminare t e di poter calcolare s. L's calcolato verrà quindi aggiunto ad una delle equazioni finali, al fine di determinare t. s e t formeranno quindi il punto di intersezione, nel caso esso esista. Nello spazio, dovremo aggiungere ancora ambedue i valori ad una terza equazione (in questo caso l'equazione con il parametro z). Se questa aggiunta fornisce un risultato, il punto di intersezione esiste. Sul piano non abbiamo bisogno di questa coordinata z, di conseguenza quest'ultimo test è superfluo. Anche sul piano potremo giungere ad una soluzione non univoca, nel caso in cui le due rette corrano parallele.

i) Rappresentazione di un piano tramite vettori:

Per i nostri obiettivi tridimensionali è particolarmente significativo poter definire anche dei piani nello spazio tramite dei vettori. L'equazione per un piano somiglia a quella di una retta come l'abbiamo vista precedentemente:

e: $\vec{p} = \vec{p}_0 + s^*\vec{a} + t^*\vec{b}$

oppure, in notazione di coordinate:

$$e: \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} p_{0x} \\ p_{0y} \\ p_{0z} \end{pmatrix} + s^* \begin{pmatrix} a_x \\ a_y y \\ a_z \end{pmatrix} + t^* \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$$

$$= \begin{pmatrix} p_{0x} + s^*a_x + t^*b_x \\ p_{0y} + s^*a_y + t^*b_y \\ p_{0z} + s^*a_z + t^*b_z \end{pmatrix}$$

Da ciò, come per le rette, potremo trarre di nuovo tre equazioni, calcolando separatamente le componenti x, y e z:

$$\text{e: } \begin{aligned} p_x &= p_{0x} + s^*a_x + t^*b_x \\ p_y &= p_{0y} + s^*a_y + t^*b_y \\ p_z &= p_{0z} + s^*a_z + t^*b_z \end{aligned}$$

Dove:

$\vec{p}$ — vettore dall'origine delle coordinate fino al punto P da calcolarsi del piano (oppure semplicemente punto P)

$\vec{p}_0$ — vettore dall'origine delle coordinate ad un punto P_0 a piacere del piano (oppure più semplicemente: punto P_0)

$\vec{a},\vec{b}$ — vettori a piacere, linearmente indipendenti, sul piano

s,t — fattori per "allungamento" (oppure riduzione) dei vettori $\vec{a}$ e $\vec{b}$, al fine di raggiungere il punto P.

"e:" significa che abbiamo a che fare con equazioni di piani. Il punto (o meglio il vettore) $\vec{p}$ rappresenta quindi tutti i punti del piano.

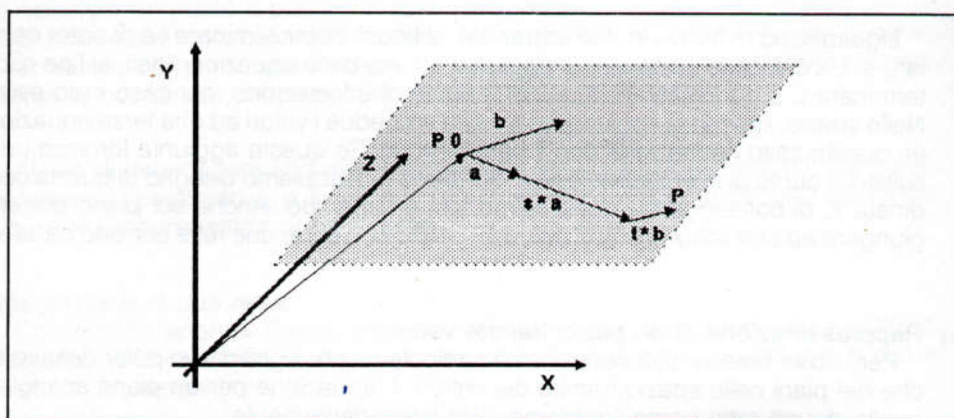


Fig. 4.15 Equazione vettoriale di una retta, tipo 1

Analogamente a quanto detto per la definizione di una retta (vedi sopra), partiamo da un punto a piacere P_0 sul piano e tracciamo due vettori (vedi addizione di vettori), che dovranno trovarsi sul piano desiderato ed essere indipendenti linearmente. "Indipendenti linearmente" (vedi sopra) significa in questo caso che i vettori non potranno essere nè paralleli nè antiparalleli (la lunghezza, invece, non ha importanza). Ambedue i vettori raggiungeranno quindi, dopo essere stati opportunamente allungati o ridotti, qualunque punto a piacere sul piano (in maniera vagamente paragonabile al cursore di un tavolo da disegno). L'allungamento oppure la riduzione potranno venire ottenuti tramite una scelta adeguata dei fattori s e t. I tre vettori risultanti $\vec{p}_0$, $s*\vec{a}$ e $t*\vec{b}$ forniranno quindi, sommati, il vettore di risultato $\vec{p}$ per il punto sul piano che stiamo cercando.

Al fine di tracciare un piano ben determinato, dovremo semplicemente fornire i parametri $\vec{p}_0$, $\vec{a}$ e $\vec{b}$. Fornendo valori a piacere per s e t, determineremo quindi tutti i punti P (oppure vettori $\vec{p}$) del piano.

Questa forma di rappresentazione di un piano, in particolare per il calcolo delle intersezioni, non sarà tuttavia molto comoda, dal momento che dovremo effettuare dei calcoli sempre con tre vettori ($\vec{p}_0$, $\vec{a}$ e $\vec{b}$) e che otterremo quasi sempre un sistema di tre equazioni in tre incognite abbastanza scomodo. Questo viene spesso sostituito da un'altra forma di equazione sul piano, che opera con la normale al piano (cioé con un vettore perpendicolare al piano). Essa è la seguente:

$$e: (P-P_0)*\vec{n} = 0$$

oppure una delle seguenti equazioni:

$$\begin{aligned} e: (P-P_0)*\vec{n} &= 0 \quad <=> \\ P*\vec{n} - P_0*\vec{n} &= 0 \quad <=> \\ P_0*\vec{n} - P*\vec{n} &= 0 \quad <=> \\ P*\vec{n} &= P_0*\vec{n} \end{aligned}$$

oppure, in notazione di coordinate:

$$e: \begin{pmatrix} p_x - p_{0x} \\ p_y - p_{0y} \\ p_z - p_{0z} \end{pmatrix} * \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = 0$$

Dal momento che in questo caso ci troviamo di fronte ad una moltiplicazione scalare, il risultato di $(P-P_0)*\vec{n}$ sarà un numero. Conformemente alla definizione del prodotto scalare, otterremo:

$$e: (px-p_{0x})*n_x + (p_y-p_{0y})*n_y + (p_z-p_{0z})*n_z = 0$$

oppure:

$$e: p_x*n_x + p_y*n_y + p_z*n_z = p_{0x}*n_x + p_{0y}*n_y + p_{0z}*n_z \quad <=>$$
$$e: p_x*n_x + p_y*n_y + p_z*n_z = d$$

dove:

P — Punto da calcolarsi del piano
P_0 — Punto fisso a piacere del piano
$\vec{n}$ — Normale al piano (vettore perpendicolare al piano)
d — Abbreviazione per $d = p_{0x}*n_x + p_{0y}*n_y + p_{0z}*n_z$

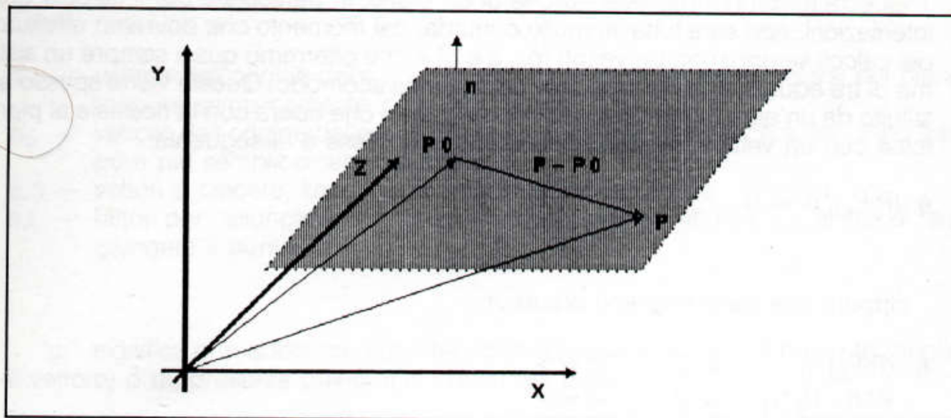


Fig. 4.16 Equazione vettoriale di una retta, tipo 2

E questo rappresenta il segreto di questa forma di piano: invece di avere a che fare con tre equazioni in forma parametrica, abbiamo a che fare con una sola. Al fine di indicare un determinato piano, avremo quindi bisogno solo dei vettori $\vec{n}$ e del punto P_0 (il prodotto scalare $P_0 * \vec{n}$ è quindi d).

Un punto P sarà quindi punto del piano solo se l'equazione è verificata: criterio ideale per la determinazione del punto di intersezione. Questa equazione tuttavia non è adeguata a calcolare un punto P del piano, dal momento che P deve già essere noto, al fine di poter costruire l'equazione stessa.

Spesso però sarà necessaria anche la vecchia forma vettoriale del piano, al fine di poter appunto determinare un punto P. Come trasformare quindi queste due equazioni l'una nell'altra?

Trasformazione: da equazione s-t a equazione della normale:
A questo scopo determiniamo dapprima il vettore della normale $\vec{n}$. Esso potrà venire determinato dal prodotto vettoriale $\vec{a} \times \vec{b}$, dal momento che si trova perpendicolare a questi due vettori della prima equazione del piano:

$$\vec{n} = \vec{a} \times \vec{b} = \begin{pmatrix} a_y * b_z - a_z * b_y \\ a_z * b_x - a_x * b_z \\ a_x * b_y - a_y * b_x \end{pmatrix}$$

E con ciò avremmo determinato $\vec{n}$. P_0 resta naturalmente uguale a P_0. s e t non sono necessari, dal momento che conosciamo la normale e il punto P.

Trasformazione: da equazione della normale a equazione s-t:

La trasformazione contraria è molto più complicata, dal momento che dovremo determinare s e t dalla normale $\vec{n}$ e dal punto P. Se i due vettori $\vec{a}$ e $\vec{b}$ non sono noti, dovremo prima di tutto calcolarli (esistono innumerevoli combinazioni per $\vec{a}$ e $\vec{b}$). Se essi invece sono noti, dovremo trasporli (la direzione e la lunghezza di questi vettori possono contenere informazioni molto importanti per determinati scopi).

Determiniamo quindi dapprima una coppia di $\vec{a}$ e $\vec{b}$ per il caso in cui questi non ci siano noti. Sappiamo che ambedue i vettori si trovano perpendicolari ad $\vec{n}$. Per i prodotti scalari vale quindi:

$\vec{a}*\vec{n} = 0$ e
$\vec{b}*\vec{n} = 0$

e con ciò:

$a_x*n_x + a_y*n_y + a_z*n_z = 0$ e
$b_x*n_x + b_y*n_y + b_z*n_z = 0$

Dal momento che per i parametri dei vettori a_x, a_y ecc. esistono infinite soluzioni (escluso $\vec{a}=\vec{b}$ oppure $\vec{a}$ = vettore zero oppure $\vec{b}$ = vettore zero), scegliamone una specifica che ci semplifichi il lavoro:

Vediamo dapprima un caso particolare:

Se due parametri di $\vec{n}$ sono uguali a zero, impostiamo uno dei parametri corrispondenti di $\vec{a}$ uguale a 1, l'altro uguale a 0. Per $\vec{b}$, scegliamo i parametri esattamente al contrario. Il parametro di $\vec{n}$, il cui valore era diverso da 0, fa in modo che i parametri corrispondenti di $\vec{a}$ e $\vec{b}$ divengano uguali a 0 (es: con $n_x=0$, $n_y=5$, $n_z=0$, impostiamo: $a_x=1$, $a_y=0$, $a_z=0$ e $b_x=0$, $b_y=0$, $b_z=1$: dal momento che $n_x=n_z=0$, avremo $a_x=b_z=1$ e $a_z=b_x=0$ dal momento che ny <> 0, avremo $a_y=b_y=0$).

Dopo aver controllato il caso specifico di cui sopra, scegliamo due parametri di $\vec{n}$ che non siano uguali a 0. Chiamiamoli n_1 ed n_2. Il terzo parametro che abbiamo tralasciato, il cui valore può essere a piacere, si chiamerà n_0. I parametri corrispondenti di $\vec{a}$ e $\vec{b}$ saranno quindi: a_1, a_2 e a_0, oppure b_1, b_2 e b_0. Quindi scegliamo:

$a_0 = b_0 = 0$
$a_2 = b_1 = 1$
$a_1 = -n_2/n_1$
$b_2 = -n_1/n_2$

Con ciò avremo determinato una coppia di $\vec{a}$ e $\vec{b}$.

Il calcolo di s e t è un po' più complicato. Anche in questo caso dovremo effettuare una differenziazione di casi. Il punto di partenza è la forma della normale del piano:

$$(P\text{-}P_0)*\vec{n} = 0$$

Per semplicità impostiamo, a partire da questo momento:

$$\vec{v} = P\text{-}P_0$$

Quindi otterremo:

$$\vec{v} * \vec{n} = 0$$

Trasformiamo la forma s-t del piano come segue:

$$\begin{aligned} P &= P_0 + s*\vec{a} + t*\vec{b} \quad &<=> \\ s*\vec{a} &= (P\text{-}P_0) - t*\vec{b} \quad &<=> \\ s*\vec{a} &= \vec{v} - t*\vec{b} \end{aligned}$$

E, in notazione di coordinate:

$$\begin{aligned} s*a_x &= v_x - t*b_x \\ s*a_y &= v_y - t*b_y \\ s*a_z &= v_z - t*b_z \end{aligned}$$

v_x, v_y e v_z sono noti esattamente come a_x, a_y, a_z, b_x, b_y e b_z. Solo le due variabili s e t debbono venire determinate. A tale scopo avremo bisogno solo di due equazioni, che sceglieremo fra le tre (i suffissi i e j stanno per x, y oppure per z):

$$\begin{aligned} s*a_i &= v_i - t*b_i \\ s*a_j &= v_j - t*b_j \end{aligned}$$

Dal momento che i vettori $\vec{a}$ e $\vec{b}$ non possono essere vettori 0, almeno un parametro di $\vec{a}$ e $\vec{b}$ dovrà essere diverso da 0), scegliamo come prima equazione quella per la quale sia ai che l'espressione $b_j*a_i - b_i*a_j$ (vedi sotto) sia diversa da 0. Ciò semplificherà molto le cose. Se non fosse possibile, avremmo commesso un errore da qualche parte. I vettori $\vec{a}$ e $\vec{b}$ non sono indipendenti linearmente. Calcoliamo:

$$\begin{aligned} s*a_i &= v_i - t*b_i \quad &<=> \\ s &= (v_i\text{-}t*b_{ji})/a_i \end{aligned}$$

E utilizziamolo nella seconda equazione:

$$\begin{aligned} (v_i - t*b_i)/a_i * a_i &= v_j - t*b_j \quad &<=> \\ v_i*a_j - t*b_i*a_j &= v_j*a_i - t*b_j*a_i \quad &<=> \\ t*(b_j*a_i - b_i*a_j) &= v_j*a_i - v_i*a_j \quad &<=> \end{aligned}$$

$$t = \frac{v_j * a_i - v_i * a_j}{b_j * a_i - b_i * a_j}$$

Abbiamo potuto effettuare anche quest'ultimo passaggio, dal momento che abbiamo controllato precedentemente che $b_j * a_i - b_i * a_j$ siano veramente diversi da 0. t viene determinato e potrà quindi venire utilizzato nella equazione per la determinazione di s. E con ciò avremmo completato l'equazione del piano.

4.4 Proiezioni - da 3 a 2

Se vogliamo rappresentare sul nostro schermo degli oggetti tridimensionali, ci troveremo di fronte al problema di come realizzare su di esso un'immagine bidimensionale, dal momento che lo schermo tridimensionale non è ancora stato inventato (anche se tecniche quali l'olografia potrebbero fornirne il fondamento teorico). Dovremo inventarci un procedimento che produca dalle coordinate tridimensionali di un singolo punto delle coordinate bidimensionali per il monitor. Si tratta quindi del problema della proiezione. Con il termine proiezione potremo immaginare la stessa cosa che accade nella proiezione del mondo tridimensionale su di una pellicola, necessariamente bidimensionale, di una macchina fotografica. Nel nostro caso, il monitor assume il ruolo della pellicola. Quello che invece accade automaticamente fuori dalla macchina fotografica, dovrà venire calcolato dal computer punto per punto.

Questa analogia con la macchina fotografica dovrà venire sempre tenuta presente nei capitoli seguenti, nel caso in cui si abbiano problemi di comprensione. In essi incontreremo infatti dei processi che, in linea di principio, mantengono la stessa analogia.

Per quanto concerne i problemi matematici, siamo ora in possesso degli strumenti necessari per le operazioni grafiche, che andremo ad apprendere nel presente capitolo, già a partire dall'ultimo capitolo. Dovremo solo trasferire tali principi al mondo tridimensionale. Tutte le matrici, sia matrici di punti che matrici di trasformazione, verranno solo ampliate.

Un punto, in un sistema di coordinate bidimensionale, potrà venire espresso anche con coordinate tridimensionali (infatti viene aggiunta solo la coordinata z). La coordinata z verrà quindi impostata a 0. Il piano del sistema di coordinate bidimensionale si trova quindi esattamente sul piano x,y del sistema tridimensionale che attraversa il punto 0. Trasformiamo quindi coordinate bidimensionali in coordinate tridimensionali come segue:

$$P(x,y) = P(x,y,0)$$

Naturalmente dovrà trattarsi dello stesso punto e di quello soltanto. Allo stesso modo trasformeremo la relativa matrice del punto P come segue:

$$P = (x\ y) = (x\ y\ 0)$$

E, in coordinate omogenee (ved. punto 3.2.5, sotto il titolo "Traslazione"):

P = (x*w y*w w) = (x*w y*w 0*w w)

Anche in questo caso la coordinata z è uguale a 0. In generale scriveremo quindi per un punto tridimensionale quanto segue:

P(x,y,z) — Notazione in coordinate
P = (x y z) — Notazione matriciale
P = (x*w y*w z*w w) — Notazione matriciale per coordinate omogenee

Allo stesso modo dovremo ampliare le matrici di trasformazione. Approfondiremo ulteriormente le formule generali per i rapporti tridimensionali. Per il momento ampliamo solo le matrici per trasformazioni da bidimensionali a tridimensionali. Tutte le matrici verranno ampliate di una riga ed una colonna. Dalla matrice per il cambio di scala:

$$S(S_x,S_y) = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

come abbiamo imparato nel capitolo dedicato alla grafica bidimensionale, otterremo:

$$S(S_x,S_y) = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

e, in coordinate omogenee:

$$S(S_x,S_y) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Allo stesso modo potremo ampliare tutte le altre matrici di trasformazione. Ecco la matrice di traslazione ampliata (potrete verificarla con i calcoli):

$$T(T_x,T_y) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & 0 & 1 \end{pmatrix}$$

Facciamo tuttavia attenzione al fatto che in questi casi si tratta sempre delle stesse trasformazioni. Ciò significa che si ipotizza che la coordinata z sia sempre uguale a 0 e che non si desideri nè ingrandire nè ridurre in direzione z.

Dopo questo chiarimento, rivolgiamoci direttamente alla proiezione, già accennata, di coordinate tridimensionali su di un piano. Esistono molte possibilità a seconda degli obiettivi che ci si è preposti. Nel presente caso esaminiamo due di questi tipi di proiezioni:

— proiezione parallela
— proiezione centrale (chiamata anche proiezione prospettica)

La prima forma è molto semplice da realizzare da un punto di vista matematico ed è particolarmente adeguata per immagini in scala, che per esempio vengono utilizzate in architettura. La seconda necessita di un tempo di calcolo maggiore. Tuttavia si tratta della rappresentazione più realistica per l'occhio umano.

4.4.1 Il sistema più semplice: proiezione parallela

Nella trasformazione di coordinate da tridimensionali a bidimensionali potrebbe anche venirci in mente di tralasciare la coordinata z, che disturba, cioè di impostarla uguale a 0 per tutti i punti in questione. Nella pratica, questo è già un caso particolare della proiezione parallela. Essa infatti viene chiamata così proprio perché da tutti i punti tridimensionali di una immagine vengono tracciate linee parallele (cioè vettori) fino al piano x,y (z=0). I punti nei quali tali vettori di proiezione incontrano il piano (lo intersecano) rappresentano i cosiddetti punti di proiezione. Per essi la coordinata z è uguale a 0 e di conseguenza può venire tralasciata, a tutto vantaggio di un sistema di coordinate semplificato.

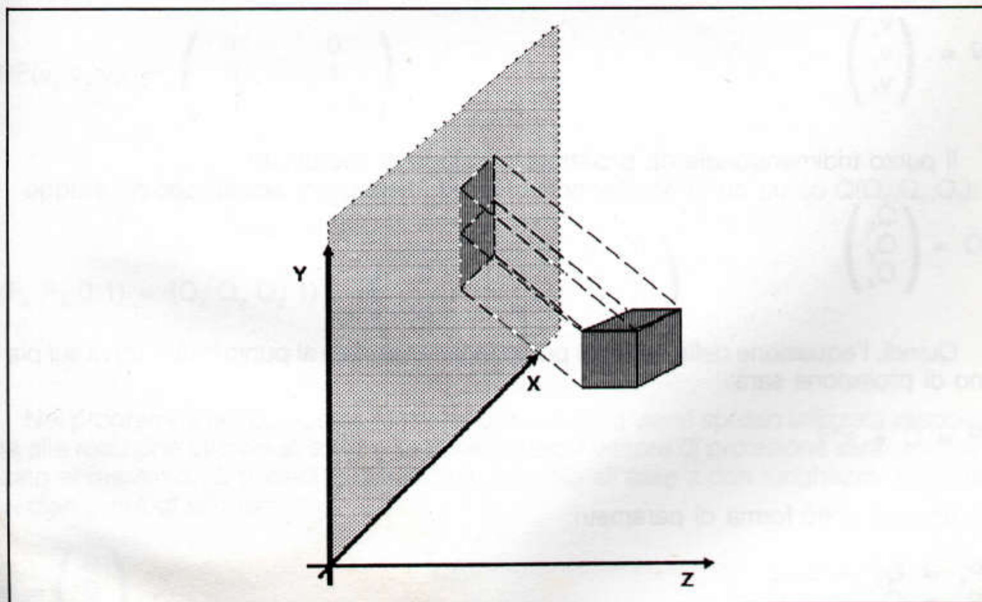


Fig. 4.17 Costruzione della proiezione parallela

Nel caso specifico appena visto, nel quale la coordinata z di ogni punto viene semplicemente impostata come uguale a 0, i vettori di proiezione corrono ancora paralleli all'asse z, per cui le coordinate x ed y non si modificano al momento della proiezione, indipendentemente dal valore della coordinata z. In questo caso si tratta di una forma della cosiddetta proiezione ortogonale, nella quale i vettori di proiezione si trovano perpendicolari (= ortogonali) al piano di proiezione.

Esistono anche altre proiezioni parallele che vengono applicate per esempio nei disegni tecnici (es: proiezione cavaliera). In essa il piano di proiezione non sempre si trova sul piano x,y. Possono venire utilizzati anche i piani xz, yz oppure altri a piacere. Le loro descrizioni matematiche saranno diverse da quelle seguenti. Spesso tuttavia si tratta di modifiche non essenziali, facili e veloci da eseguire.

Il seguente procedimento ipotizza che il piano xy rappresenti il piano di proiezione. I vettori di proiezione, si tratta a dir la verità di un solo vettore di proiezione, dal momento che è solo la sua direzione ad avere importanza, saranno a piacere, sempre che non siano paralleli al piano.

Cerchiamo di ricordare la forma vettoriale dell'equazione di una retta:

$$\vec{P} = \vec{P}_0 + s^*\vec{a}$$

Ipotizziamo che il nostro vettore di proiezione v (cioè il vettore di direzione di tutte le linee dal punto tridimensionale al piano di proiezione) sia:

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Il punto tridimensionale da proiettare possiede le coordinate:

$$\vec{Q} = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \end{pmatrix}$$

Quindi, l'equazione della retta dal punto tridimensionale al punto che si trova sul piano di proiezione sarà:

$$\vec{P} = \vec{Q} + s^*\vec{v}$$

oppure, sotto forma di parametri:

$$P_x = Q_x + s^*v_x$$
$$P_y = Q_y + s^*v_y$$
$$P_z = Q_z + s^*v_z$$

P è quindi il punto ricercato sul piano di proiezione, cioè il punto nel quale la retta taglia il piano. A questo punto dovremo ancora determinare quale dovrà essere il valore di s. Dopo ciò, potremo determinare P. Il nostro piano, come determinato precedentemente, dovrà essere il piano xy. La coordinata z del punto di proiezione sul piano sarà quindi uguale a 0. Se però Pz è uguale a 0, otterremo dalla terza equazione di parametri:

$$0 = Q_z + s * v_z \quad <=>$$

$$s = -\frac{Q_z}{v_z}$$

Ciò potrà venire di nuovo utilizzato nelle prime due equazioni, e dopo un paio di sostituzioni, otterremo la formula per la proiezione parallela:

$$P_x = Q_x - Q_z *(v_x/v_z)$$
$$P_y = Q_y - Q_z *(v_y/v_z)$$

dove:

P_x, P_y — Coordinate del punto del piano proiettato
Q_x, Q_y, Q_z — Coordinate del punto nello spazio da proiettare
V_x, V_y, V_z — Vettore di proiezione

Da ciò, determiniamo la seguente matrice di trasformazione:

$$PP(v_x,v_y,v_z) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -v_x/v_z & -v_y/v_z \end{pmatrix}$$

oppure, in coordinate omogenee con la trasformazione di un punto $Q(Q_x,Q_y,Q_z)$:

$$(P_x\ P_y\ 0\ 1) = (Q_x\ Q_y\ Q_z\ 1) * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -v_x/v_z & -v_y/v_z & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Nei programmi per computer, la proiezione parallela viene spesso utilizzata associata alla rotazione attorno all'asse x. In questo caso il vettore di proiezione sarà semplificato al massimo. Si tratterà quasi di una parallela all'asse z con lunghezza 1, come vediamo qui di seguito:

$$\vec{v} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

La matrice di trasformazione PP(0,0,-1) viene con ciò molto semplificata (provate a controllare). La conseguenza è che incontreremo il caso particolare nel quale vengono utilizzate le coordinate x ed y, mentre la coordinata z sparirà (vedi sopra). Prima della proiezione, l'intera immagine sarà ruotata attorno all'asse x e all'asse y (sarebbe naturalmente è anche possibile una rotazione attorno all'asse z). Con ciò la coordinata z originale avrà di nuovo un influsso sulle coordinate di proiezione risultanti. L'angolo di rotazione e le sue funzioni angolari verranno calcolati una volta sola all'inizio della proiezione e in seguito trasferiti nel calcolo come costanti. Il vantaggio di questo metodo è che gli angoli di rotazione rappresentano praticamente l'angolo di visuale, con il quale l'osservatore sta guardando l'immagine. Nei programmi, gli angoli sono molto più semplici da maneggiare dei vettori di proiezione. Dal momento che ci occuperemo solo in seguito di rotazioni nello spazio, vediamo una matrice di rotazione e di proiezione parallela:

$$\vec{P} = \vec{Q} * R_y(b) * R_x(a) * PP(0,0,-1) =$$

$$(P_x\ P_y\ 0\ 1) =$$

$$(Q_x\ Q_y\ Q_z\ 1) * \begin{pmatrix} \cos(b) & \sin(a)*\sin(b) & 0 & 0 \\ 0 & \cos(a) & 0 & 0 \\ \sin(b) & -\sin(a)*\cos(b) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

e, in forma di parametri:

$$P_x = Q_x*\cos(b) + Q_z*\sin(b)$$
$$P_y = Q_x*\sin(a)*\sin(b) + Q_y*\cos(a) - Q_z*\sin(a)*\cos(b)$$

dove:

$P\ (P_x,P_y)$	—	Punto risultante nel piano
$Q\ (Q_x,Q_y,Q_z)$	—	Punto di partenza nello spazio
$R_y(b)$		Matrice di rotazione: rotazione attorno all'asse y con l'angolo b
$R_x(a)$	—	Matrice di rotazione: rotazione attorno all'asse x con l'angolo a
PP(0,0,-1)	—	Matrice di trasformazione: proiezione parallela con vettore di proiezione $\vec{v}$ = (0 0 -1)
a	—	Angolo di rotazione attorno all'asse x
b	—	Angolo di rotazione attorno all'asse y

La rotazione degli oggetti può venire vista anche come rotazione dei piani di proiezione, la quale naturalmente avrà luogo in direzione contraria.

La forma parametrica indicata è esattamente la formula che utilizzeremo nel programma seguente, al fine di proiettare un oggetto tridimensionale sul monitor. Osserviamo con calma ed attenzione tale programma: per la prima volta vedremo come trasferire nella pratica tutta quella grigia teoria cui finora ci siamo dedicati.

```
'*************************
'**     Proiezione       **
'**     Parallela        **
'**                      **
'*************************

PI = 3.141593

'Traslazione delle coordinate del piano:
'(Posizione dell'origine delle coordinate)
maus% = MOUSE(0)
v1    = MOUSE(3)   'Prelevamento delle
v2    = MOUSE(4)   'coordinate del Mouse

'Angolo di rotazione di Start del piano di proiezione (Gradi):
ag = 10              'Rotazione attorno all'asse x
bg = -10             'Rotazione attorno all'asse y

a = PI * ag/180    'Trasformazione in gradi
b = PI * bg/180

'Valore incremento di rotazione (Gradi):
dig = 5

di = PI * dig/180  'Trasformazione in RAD

'Apertura di un nuovo Schermo con Finestra:
SCREEN 2,320,200,2,1
WINDOW 2,"Proiezione Parallela trid.",(0,10)-(297,186),1+2+8,2

PALETTE 0,0,0,0      'Sfondo: nero
PALETTE 2,1,.6,.67   'Colore 1: Rosso
PALETTE 3,1,1,.13    'Colore 2: Giallo
PALETTE 1,.4,.6,1    'Colore 3: Blu
COLOR ,0             'Colore dello sfondo = 0

'Lettura dati oggetto nelle matrici:
'xr%(), yr%(), zr%() - Coordinate dei punti spaziali
'ls%(), le%()        - Collegamenti Linee
get.objects


'Loop principale:
'***************

flag%=0
WHILE flag%<>1
```

```
  projektion        'Proiezione di tutti i punti

  CLS               'Cancellazione Finestra

  'Tracciato dell'asse delle coordinate:
  PATTERN &HAAAA    'Motivo di righe: a strisce
  COLOR 2
  FOR i=0 TO 2
    LINE (xe%(ls%(i)),ye%(ls%(i)))-(xe%(le%(i)),ye%(le%(i)))
  NEXT i

  PATTERN &HFFFF    'Motivo di righe = a zig-zag
  COLOR 3
  'Output delle linee (Tracciatura dell'oggetto)
  FOR i=3 TO al%-1
    LINE (xe%(ls%(i)),ye%(ls%(i)))-(xe%(le%(i)),ye%(le%(i)))

NEXT i

maus% = 0 : flag%=0
WHILE maus%<>1 AND flag%=0
  SLEEP             'Attesa di un evento

  'Adozione delle Coordinate del Mouse come
  'nuova origine delle coordinate:
  maus%=MOUSE(0)
  IF maus%=1 THEN
    v1=MOUSE(3)
    v2=MOUSE(4)
  END IF

  c% = ASC(INKEY$+CHR$(0))
  IF c%=31 THEN  'Cursore a sinistra =>
    b=b+di       'Aumento dell'asse y
    flag% = -1   'Flag per nuovo disegno
  END IF
  IF c%=30 THEN  'Cursore a destra =>
    b=b-di       'diminuzione angolo rotazione attorno all'asse y
    flag% = -1   'Flag per nuovo disegno
  END IF
  IF c%=28 THEN  'Cursore in alto =>
    a=a+di       'aumento angolo rotazione attorno all'asse x
    flag% = -1   'Flag per nuovo disegno
  END IF
```

```
    IF c%=29 THEN   'Cursore verso il basso =>
      a=a-di        'diminuzione angolo rotazione attorno all'asse x
      flag% = -1    'Flag per nuovo disegno
    END IF

    'Se la finestra e' chiusa -> Fine
    IF WINDOW(7)=0 THEN
      flag%=1
    END IF

  WEND
WEND

WINDOW OUTPUT 1
SCREEN CLOSE 2

END


'Lettura dati oggetto:

SUB get.objects STATIC

  SHARED ap%

  READ ap%
  DIM SHARED xr%(ap%),yr%(ap%),zr%(ap%)
  DIM SHARED xe%(ap%),ye%(ap%)

  FOR i=0 TO ap%-1
    READ xr%(i),yr%(i),zr%(i)
  NEXT i

  'Lettura definizione linea:
  SHARED al%

  READ al%
  DIM SHARED ls%(al%),le%(al%)

  FOR i=0 TO al%-1
    READ ls%(i),le%(i)
  NEXT i

END SUB

'Proiezione di tutte le coordinate spaziali su di un piano:
SUB projektion STATIC
  SHARED ap%
  SHARED a,b,v1,v2
```

```
  'Calcolo delle costanti per la proiezione:
  si.a=SIN(a)
  co.a=COS(a)
  si.b=SIN(b)
  co.b=COS(b)

  FOR i=0 TO ap%-1

    Qx = xr%(i)
    Qy = yr%(i)
    Qz = -zr%(i)      'trasformazione in un sistema sinistro

    xe%(i) = v1 + Qx*co.b + Qz*si.b
    ye%(i) = v2 - (Qx*si.a*si.b + Qy*co.a - Qz*si.a*co.b)

  NEXT i
END SUB


'Dati oggetto:
'************

'Coordinate tridimensionali del punto:

'Numero dei punti:
DATA 36

'Asse coordinate:
DATA -10,  0,  0,    80, 0,   0
DATA   0,-10,  0,     0,90,   0
DATA   0,  0,-10,     0, 0, 180

'Punti oggetto:
DATA  5,10, 5,    5,45, 5,   15,65, 5
DATA 25,45, 5,   25,10, 5,    5,10,55
DATA  5,45,55,   15,65,55,   25,45,55
DATA 25,10,55,   13,10, 5,   13,20, 5
DATA 17,20, 5,   17,10, 5,    7,33, 5
DATA  7,38, 5,   12,38, 5,   12,33, 5
DATA 18,33, 5,   18,38, 5,   23,38, 5
DATA 23,33, 5,   25,33,12,   25,38,12
DATA 25,38,23,   25,33,23,   25,33,32
DATA 25,38,32,   25,38,43,   25,33,43


'Collegamenti linee:

'Numero delle linee:
DATA 40
```

```
'Asse coordinate:
DATA  0, 1,    2, 3,    4, 5

'Linee Oggetto:
DATA  6, 7,    7, 8,    8, 9,    9,10
DATA 10, 6,    7, 9,   11,12,   12,13
DATA 13,14,   14,15,   15,11,   12,14
DATA  6,11,    7,12,    8,13,    9,14
DATA 10,15,   16,17,   17,18,   18,19
DATA 19,16,   20,21,   21,22,   22,23
DATA 23,20,   24,25,   25,26,   26,27
DATA 27,24,   28,29,   29,30,   30,31
DATA 31,28,   32,33,   33,34,   34,35
DATA 35,32
```

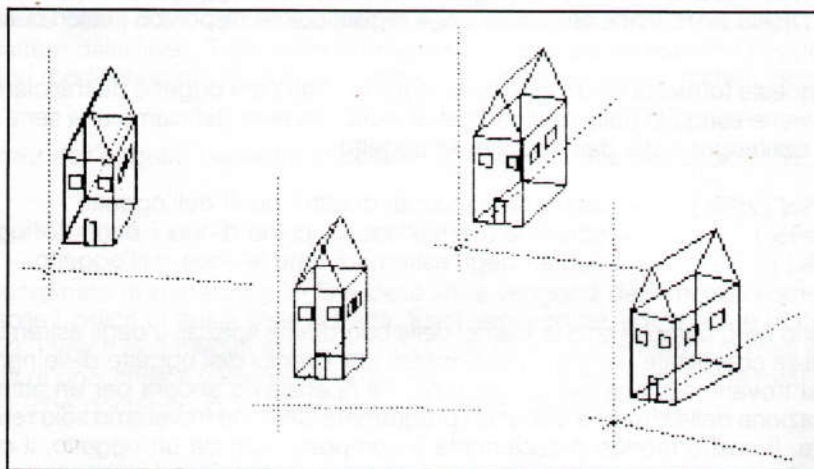


Fig. 4.18 Proiezione parallela

Ecco finalmente il nostro primo programma tridimensionale; mandandolo in esecuzione ci meraviglieremo di quello che possiamo ottenere con dei mezzi così semplici. Sullo schermo appare una casa ed un sistema di coordinate: spostando il puntatore con il mouse e premendo il bottone la casa ed il sistema di coordinate verranno cancellati e ridisegnati nella nuova posizione, mentre premendo un tasto cursore la casa si metterà a ruotare.

Ciò che abbiamo visto sul monitor è il risultato di quanto abbiamo appreso finora. Vediamo quindi un elenco delle possibilità di manipolazione con questo programma:

tasto sinistro del mouse — posizionamento del sistema di coordinate tridimensionale
cursore sinistra/destra — rotazione del piano di proiezione attorno all'asse y
cursore verso alto/basso — rotazione del piano di proiezione attorno all'asse x

Sarà ora possibile osservare la casa disegnata da tutti i lati, ruotando il piano di proiezione tramite i tasti del cursore. Se, al momento dell'avviamento del programma, l'oggetto si troverà al di fuori della finestra, posizioniamolo correttamente tramite il tasto del mouse.

Osserviamo meglio come è possibile che il programma riesca a creare tutto ciò sullo schermo. Prima di tutto, come sempre, vengono assegnate alcune variabili di base. In questo caso si tratta delle coordinate del sistema tridimensionale (v1 e v2), dei valori di partenza per la rotazione e dei due assi di rotazione (a e b) nonché il valore di, del quale gli angoli di rotazione dovranno venire incrementati o diminuiti quando si aziona un tasto cursore. Naturalmente sono possibili tutte le modifiche immaginabili. Sarà necessario fare attenzione al fatto che tutti gli angoli devono essere forniti in radianti. Questo è il motivo per cui questo programma trasforma le indicazioni in gradi in radianti.

Si predispone un nuovo schermo (risoluzione: 320 × 200, 2 bit-plane = 4 colori) e in esso viene aperta una finestra con un gadget di chiusura, un gadget di dimensionamento e una DRAG BAR. Infine attribuiamo ai 4 registri colore disponibili i valori desiderati.

Dopo queste formalità sarà necessario leggere i dati per l'oggetto da tracciare. Tale compito viene eseguito dalla routine "get.objects". In essa definiamo una serie di matrici che contengono dei dati specifici all'oggetto:

xr%( ),yr%( ),zr%( )	— coordinate spaziali di tutti i punti dell'oggetto
xe%( ),ye%( )	— coordinate proiettate sul piano di tutti i punti dell'oggetto
ls%( ),le%( )	— numeri degli estremi di tutte le linee dell'oggetto.

Vengono però riempite solo le matrici delle coordinate spaziali e degli estremi. Nelle matrici delle coordinate vengono memorizzati tutti i vertici dell'oggetto dalle righe DATA che si trovano alla fine del programma. Se ripensiamo ancora per un attimo alla organizzazione delle strutture dati in un programma CAD, ne troveremo solo realizzata una parte. Il nostro mondo praticamente è composto solo da un oggetto, il quale a sua volta è composto da molte linee (per il momento abbiamo tralasciato le superfici). Le linee, a loro volta, vengono definite dai loro estremi. 36 punti sono sufficienti per memorizzare come una casa, vertice per vertice . I 40 collegamenti tra questi punti vengono memorizzati nel secondo gruppo di righe DATA. In esso, per ogni linea, vengono indicati due valori: il numero del punto di partenza e quello del punto di arrivo. Il modo in cui sviluppare tale immagine sulla carta, di leggere sistematicamente i vari punti nonché collegarli tra loro, verrà illustrato alcuni paragrafi più avanti. Nel caso in cui si vogliano tracciare più oggetti, sarà necessario ampliare le matrici di punti e linee, finora monodimensionali, con una dimensione aggiuntiva. Essa indicherà a quale oggetto un punto o una linea appartengano.

Continuiamo ad occuparci del programma. I punti e le linee si trovano quindi nelle matrici corrispondenti e la routine "get.objects" ritorna al programma principale. Quindi troviamo un loop WHILE...WEND, nel quale ha luogo l'intero processo di input e di disegno. Tuttavia, non c'è ancora nulla da disegnare. I punti del nostro oggetto sono presenti per il momento solo in forma tridimensionale, quella che manca è la proiezione su di un piano.

Otterremo tale proiezione grazie alla routine "projektion". Ripensiamo per un attimo all'inizio del presente capitolo: ciò che viene fatto qui non è altro che l'applicazione della formula per la proiezione parallela con la rotazione dei piani di proiezione attorno agli assi x ed y. Anche i nomi delle variabili sono identici a quelli visti appunto all'inizio del presente capitolo.

Effettuiamo mediante un loop una proiezione punto per punto (il numero dei punti si trova nella variabile ap%) sul piano, e memorizziamo i risultati nelle matrici xe%() e ye%(). Le variabili di traslazione v1 e v2 riguardano lo spostamento semplice sul piano, come abbiamo già visto. Il ritorno nel programma principale ha luogo al termine di tale ciclo.

Dal momento che ora conosciamo le coordinate sul piano di ogni singolo punto, disponibili in due matrici, potremo procedere a tracciare tutto ciò sullo schermo. Su di esso, come prima cosa, verrà tracciato il sistema di coordinate, con modifica dei colori e del pattern delle linee. Tracciamo in un loop ciascuno dei tre assi. Per l'oggetto vero e proprio modificheremo di nuovo il colore e normalizzeremo il pattern delle linee.

Le linee dell'oggetto verranno impostate con un normale comando LINE:

```
LINE (xe%(ls%(i)),ye%(ls%(i)))-(xe%(le%(i)),ye%(i)))
```

Le coordinate di partenza e di fine delle linee vengono determinate come segue: la variabile i indica di quale linea si tratta. Essa serve come indice per le matrici ls%() ed le%(). I valori determinati in questo modo saranno, come noto, i numeri dei punti terminali di ogni linea e verranno a loro volta utilizzati come indici per le matrici xe%() e ye%(), che contengono le coordinate del piano desiderate.

Il resto del programma è molto semplice da comprendere. In un loop interno "WHILE...WEND" il programma attende un evento (SLEEP) e favorisce, in tale periodo, il multitasking. Se un evento ha luogo, dovrà venire determinato di quale tipo si tratta. Purtroppo l'AmigaBasic non offre una funzione compatta ed elegante come quella del C. Dovremo quindi esaminare tutte le possibilità che ci interessano ed agire di conseguenza. Tutto ciò è sufficientemente documentato dal programma. Un'ulteriore annotazione: se si seleziona il gadget di chiusura, l'AmigaBasic chiuderà automaticamente la finestra, senza comunicare ciò al programma in Basic oppure senza attendere una conferma. Per questo motivo sarà necessario utilizzare un piccolo trucco. Non appena l'ultima finestra dello schermo viene chiusa, il contenuto di WINDOW(7) diventa uguale a 0 (puntatore alla struttura Window della finestra attuale non più disponibile). Agiamo quindi di conseguenza.

E' ora giunto il momento di approfondire in che modo realizzare a tavolino un proprio oggetto e determinare le coordinate dei punti. Questo programma è particolarmente adeguato a tracciare un oggetto comunque complesso senza necessità di essere modificato, ma creando altre righe DATA.

Prendiamo l'esempio della nostra piccola casa. Il problema è ora che per noi è molto difficile determinare coordinate tridimensionali da un oggetto proiettato su di un piano. Per questo motivo dobbiamo ridurre il problema, producendo semplicemente più viste dell'oggetto desiderato. Il numero di queste viste dipende dalla complessità del singolo oggetto e dalla capacità di immaginazione spaziale del lettore. Nel caso della nostra semplice casetta, sarebbero sufficienti due viste (una frontale e una dal lato delle finestre). L'ottimale sarebbe tuttavia disporre di quattro viste, una per ogni lato (vedi figura 4.19).

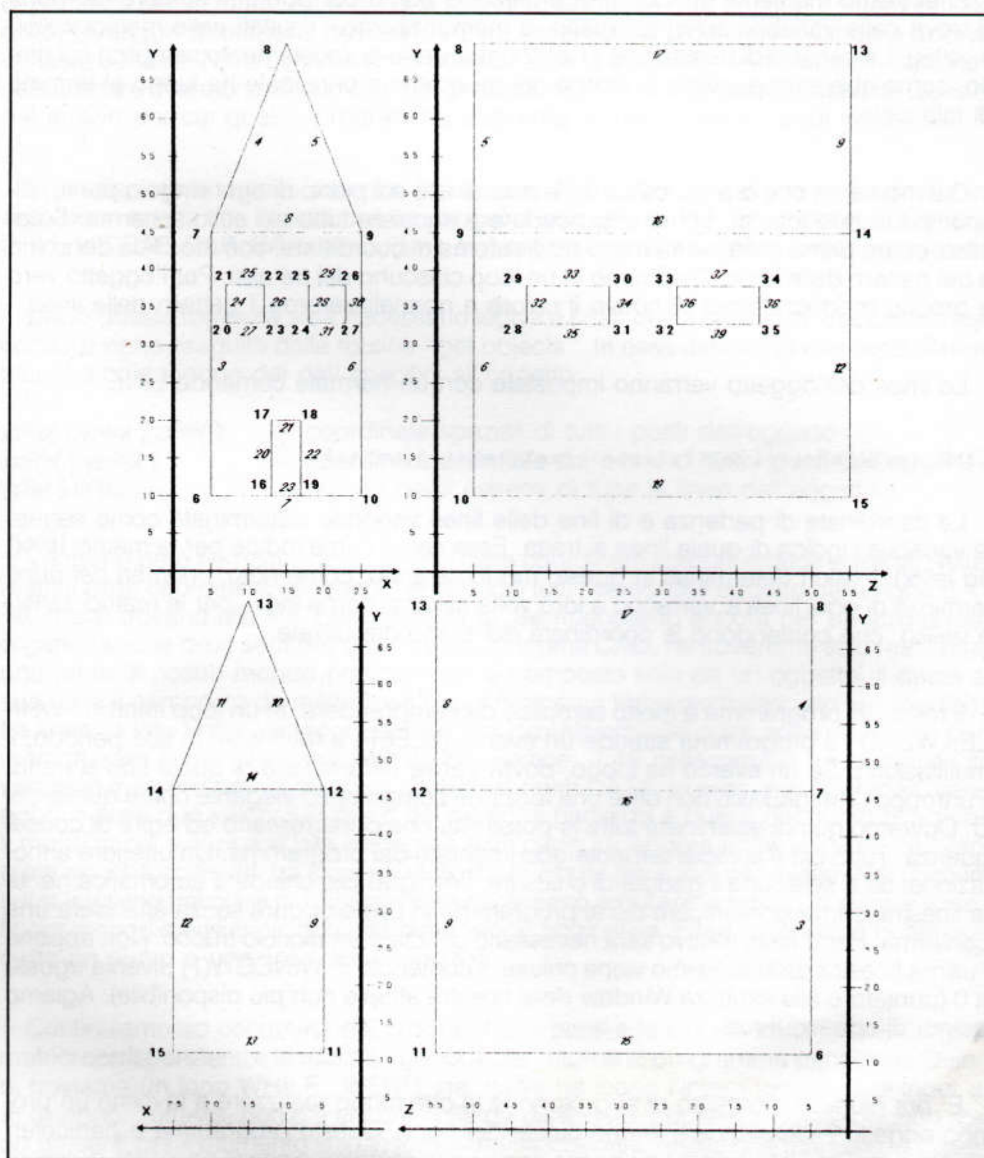


Fig. 4.19 Quattro viste della casa

Ogni vista viene presa frontalmente (quindi nessuna osservazione da un determinato angolo, cioè piano di proiezione e vettore di proiezione paralleli agli assi). Contemporaneamente dovremo tracciare anche un sistema di coordinate (xy, xz oppure yz). In caso di più viste dovremo inoltre fare attenzione al fatto che le unità di tutti gli assi aventi lo stesso nome e la loro posizione rispetto all'oggetto coincidano. Il passo successivo è la numerazione di tutti gli estremi delle linee, cominciando da 0 (nel nostro programma il sistema di coordinate viene tracciato contemporaneamente). In questo caso quindi dovremo cominciare da 6). In conclusione, numeriamo allo stesso modo le molte linee diverse (possibilmente con un altro colore). (Nel nostro esempio cominciamo con 3, dal momento che si deve tenere conto anche dei tre assi delle coordinate). In seguito ci occuperemo anche delle superfici. Cerchiamo comunque di fare attenzione ad attribuire lo stesso numero a due punti identici in viste diverse. Diversamente, ci troveremo di fronte a vertici in sovrappiù.

Creiamo quindi due tabelle. La prima contiene tutti i punti con le loro coordinate tridimensionali. La seconda conterrà tutte le linee. In essa sono contenuti i numeri di quei punti che formano una linea. Ogni punto potrà quindi venire rappresentato con la frequenza desiderata.

Le tabelle per la nostra casetta potrebbero quindi essere le seguenti:

Tabella dei punti:

Nr	x	y	z	Nr	x	y	z	Nr	x	y	z
6	5	10	5	10	13	10	5	26	23	38	5
7	5	45	5	17	13	20	5	27	23	33	5
8	15	65	5	18	17	20	5	28	25	33	12
9	25	45	5	19	17	10	5	29	25	38	12
10	25	10	5	20	7	33	5	30	25	38	23
11	5	10	55	21	7	38	5	31	25	33	23
12	5	45	55	22	12	38	5	32	25	33	32
13	15	65	55	23	12	33	5	33	25	38	32
14	25	45	55	24	18	33	5	34	25	38	43
15	25	10	55	25	18	38	5	35	25	33	43

Tabella delle linee:

Nr	P1	P2	Nr	P1	P2	Nr	P1	P2	Nr	P1	P2
3	6	7	13	15	11	23	19	16	33	29	30
4	7	8	14	12	14	24	20	21	34	30	31
5	8	9	15	6	11	25	21	22	35	31	28
6	9	10	16	7	12	26	22	23	36	32	33
7	10	6	17	8	13	27	23	20	37	33	34
8	7	9	19	9	14	28	24	25	38	34	35
9	11	12	19	10	15	29	25	26	39	35	32
10	12	13	20	16	17	30	26	27			
11	13	14	21	17	18	31	27	24			
12	14	15	22	18	19	32	28	29			

Paragoniamo queste tabelle con le righe DATA del programma precedente. Se si sono sviluppati spesso oggetti di questo genere, si starà già cercando un editor tridimensionale adeguato, con il quale poter manipolare in maniera più semplice questa mole di dati. Cerchiamo tuttavia di guardare più avanti, affinché ne valga la pena.

Un'ultima nota: cerchiamo di inserire nel programma precedente le seguenti righe DATA (non dimenticando i dati per il sistema di coordinate):

```
' Punti:
' ******
'
' Numero punti:
DATA 170
'
' Coordinate:
DATA   1, -7, 12,    1,  5, 12,    1,  5, 10,    1,  7, 10,    1,  7,  4
DATA   1,  5,  4,    1,  5,  2,    1,  7,  2,    1,  6,  9,    1,  2,  9
DATA   1,  2,  5,    1,  6,  5,    1,  6,  6,    1,  6,  7,    1,  6,  8
DATA   1,  2,  6,    1,  2,  7,    1,  2,  8,    1,  4, 14,    1,  3, 14
'
DATA   1,  3, 13,    1,  4, 13,    1,  4,  1,    1,  3,  1,    1,  3,  0
DATA   1,  4,  0,    1,  8, 14,    1,  2, 14,    1,  2,  0,    1,  8,  0
DATA  14, 12,  3,   14, 12, 11,    2,  8,  7,    2,  9,  6,    2, 10,  7
DATA   2,  9,  8,    8,  9,  1,    8,  9, 13,    8,  9, 11,   10, 10,  8
'
DATA   8,  9,  6,   10, 10,  3,    6,  8, 13,    8, 13, 13,    4,  2, 14
DATA   4,  2,  0,    4,  5, 14,    4,  5,  0,   10,  5, 14,   10,  5,  0
DATA  10,  2, 14,   10,  2,  0,   22,  2, 14,   22,  2,  0,   22,  5, 14
DATA  22,  5,  0,   28,  5, 14,   28,  5,  0,   28,  2, 14,   28,  2,  0
'
DATA  31,  2, 14,   31,  2,  0,   31,  2,  3,   32,  2,  3,   31,  6, 14
DATA  31,  6,  0,   27, 12, 11,   27, 12,  3,    5,  3, 13,    5,  3, 10
DATA   5,  3,  4,    5,  3,  1,    5,  1, 13,    5,  1, 10,    5,  1,  4
DATA   5,  1,  1,    6,  0, 13,    6,  0, 10,    6,  0,  4,    6,  0,  1
'
DATA  10,  8, 14,   10,  8,  0,   19, 12, 11,   19, 12,  3,    8,  0, 13
DATA   8,  0, 10,    8,  0,  4,    8,  0,  1,    9,  1, 13,    9,  1, 10
DATA   9,  1,  4,    9,  1,  1,    9,  3, 13,    9,  3, 10,    9,  3,  4
DATA   9,  3,  1,    8,  4, 13,    8,  4, 10,    8,  4,  4,    8,  4,  1
'
DATA   6,  4, 13,    6,  4, 10,    6,  4,  4,    6,  4,  1,   23,  3, 13
DATA  23,  3, 10,   23,  3,  4,   23,  3,  1,   23,  1, 13,   23,  1, 10
DATA  23,  1,  4,   23,  1,  1,   24,  0, 13,   24,  0, 10,   24,  0,  4
DATA  24,  0,  1,   19,  2, 14,   19,  2,  0,   19,  7, 14,   19,  7,  0
'
DATA  26,  0, 13,   26,  0, 10,   26,  0,  4,   26,  0,  1,   27,  1, 13
DATA  27,  1, 10,   27,  1,  4,   27,  1,  1,   27,  3, 13,   27,  3, 10
DATA  27,  3,  4,   27,  3,  1,   26,  4, 13,   26,  4, 10,   26,  4,  4
DATA  26,  4,  1,   24,  4, 13,   24,  4, 10,   24,  4,  4,   24,  4,  1
'
```

```
DATA   7,  2, 13,    7,  2,  1,   25,  2, 13,   25,  2,  1,   18,  7, 14
DATA  18,  7,  0,   18,  6, 14,   18,  6,  0,   15, 12, 11,   15, 12,  3
DATA '18, 12, 11,   18, 12,  3,   18,  8, 14,   18,  8,  0,   20, 12, 11
DATA  20, 12,  3,   20,  8, 14,   20,  8,  0,   28,  8, 14,   28,  8,  0
'
DATA  26, 12, 11,   26, 12,  3,   19,  8, 14,   19,  8,  0,   28, 10, 11
DATA  30,  7, 12,   30,  7,  2,   28, 10,  4,    6,  8, 14,    6,  8,  0
'
'
' Linee:
' ******
'
' Numero Linee:
DATA 192
'
' Collegamenti:
DATA   0,  1,    1,  2,    2,  3,    3,  0,    4,  5
DATA   5,  6,    6,  7,    7,  4,    8,  9,    9, 10
DATA  10, 11,   11,  8,   12, 15,   13, 16,   14, 17
DATA  18, 19,   19, 20,   20, 21,   21, 18,   22, 23
'
DATA  23, 24,   24, 25,   25, 22,   26, 27,   27, 28
DATA  28, 29,   29, 26,  158, 64,   30, 31,  159, 65
DATA  32, 34,   33, 35,   36, 37,   38, 39,   40, 41
DATA  42, 43,   27, 44,   44, 46,   46, 48,   48, 50
'
DATA  50, 52,   52, 54,   54, 56,   56, 58,   58, 60
DATA  60, 64,   64, 66,   66, 31,   31,168,  168, 26
DATA  28, 45,   45, 47,   47, 49,   49, 51,   51, 53
DATA  53, 55,   55, 57,   57, 59,   59, 61,   61, 65
'
DATA  65, 67,   67, 30,   30,169,  169, 29,   66, 67
DATA  64, 65,   60, 61,   68, 72,   72, 76,   76, 84
DATA  84, 88,   88, 92,   92, 96,   96,100,  100, 68
DATA  69, 73,   73, 77,   77, 85,   85, 89,   89, 93
'
DATA  93, 97,   97,101,  101, 69,   70, 74,   74, 78
DATA  78, 86,   86, 90,   90, 94,   94, 98,   98,102
DATA 102, 70,   71, 75,   75, 79,   79, 87,   87, 91
DATA  91, 95,   95, 99,   99,103,  103, 71,  140,141
'
DATA 104,108,  108,112,  112,120,  120,124,  124,128
DATA 128,132,  132,136,  136,104,  105,109,  109,113
DATA 113,121,  121,125,  125,129,  129,133,  133,137
DATA 137,105,  106,110,  110,114,  114,122,  122,126
'
DATA 126,130,  130,134,  134,138,  138,106,  107,111
DATA 111,115,  115,123,  123,127,  127,131,  131,135
DATA 135,139,  139,107,  142,143,   48, 80,   80,148
DATA 150,152,  152, 80,   82,162,  162,116,  118,144
'
```

```
DATA 144,146,  154,156,  156,158,  158,160,   49, 81
DATA  81,149,  151,153,  153, 81,   83,163,  163,117
DATA 119,145,  145,147,  155,157,  157,159,  159,161
DATA  62, 63,  164,165,  165,166,  166,167,  167,164
'
DATA  68, 69,   70, 71,   72, 73,   74, 75,   76, 77
DATA  78, 79,   84, 85,   86, 87,   88, 89,   90, 91
DATA  92, 93,   94, 95,   96, 97,   98, 99,  100,101
DATA 102,103,  104,105,  106,107,  108,109,  110,111
'
DATA 112,113,  114,115,  120,121,  122,123,  124,125
DATA 126,127,  128,129,  130,131,  132,133,  134,135
DATA 136,137,  138,139
```

4.4.2 Proiezione centrale

Anche se quanto realizzato finora ci entusiasmerà, rimangono tuttavia alcune questioni da eliminare. Eccone una da migliorare.

Se osserviamo attentamente l'ultimo programma, noteremo immediatamente che la nostra casetta appare più grande verso il fondo, benché la parete posteriore sia esattamente uguale a quella anteriore. E' chiaro che si tratta di un'illusione ottica, dal momento che siamo abituati a vedere degli oggetti sempre più piccoli, man mano che ci allontaniamo da essi. Un esempio classico sono i binari ferroviari, i quali, notoriamente paralleli, sembrano incontrarsi all'orizzonte. Tale fenomeno viene chiamato anche deviazione prospettica. Al momento della proiezione parallela, tale fenomeno non è stato tenuto in debita considerazione. Il nostro occhio (o meglio, il nostro cervello) è abituato all'effetto prospettico e ne trae la conclusione contraria: se un oggetto (la parete posteriore della casa) che si trova dietro un altro oggetto (la parete anteriore) ne possiede la stessa dimensione, in realtà esso sarà più grande di quello anteriore. Questo è il motivo per cui la parete anteriore ci sembra più grande.

Come eliminare questo difetto? Una tecnica molto diffusa è l'introduzione del punto di fuga (alcuni pittori inseriscono anche più di un punto di fuga nelle loro opere, per farle apparire più realistiche. Si tratta comunque di effetti che possono venire ottenuti matematica, anche con un solo punto di fuga).

I lati allungati di tutti gli oggetti che noi osserviamo nella natura, sembrano incontrarsi in lontananza in un solo punto, il punto di fuga (chiamato anche centro prospettico). Ora è nostra intenzione tenere conto di tale punto di fuga al momento della proiezione delle coordinate spaziali su di un piano. Ciò accade con la cosiddetta proiezione centrale o prospettica.

In ogni caso, il nostro punto di fuga si trova da un'altra parte. Esso rappresenta la posizione nella quale si trova l'osservatore oppure il suo occhio (o la macchina fotografica), cioè la posizione nella quale ricadono tutti i raggi luminosi dell'oggetto osservato (vedi fig. 4.20).

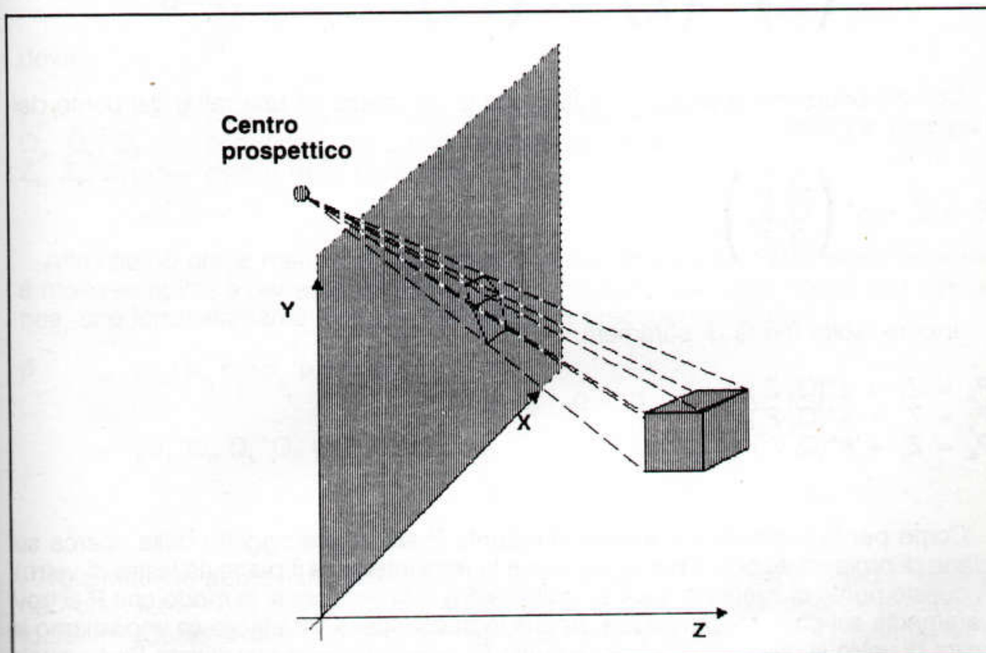


Fig. 4.20 Proiezione centrale

Immaginiamo di guardare attraverso una lastra di vetro opaca. Su tale lastra sarà quindi presente, anche se lattiginosa, una immagine del mondo circostante. I raggi luminosi partono quindi dall'oggetto stesso, per poi attraversare la lastra di vetro e giungere infine al nostro occhio.

La lastra di vetro rappresenta la pellicola oppure il nostro schermo. A noi interessa sapere in quale posizione i raggi incontrano la lastra di vetro. Ci troviamo di fronte allo stesso problema incontrato nel caso della proiezione parallela. La differenza è che ora i raggi non sono paralleli, come lo erano in tale caso, ma si incontreranno in un punto.

Ipotizziamo che il centro della proiezione (cioè il punto di fuga) si trovi nel punto $Z(Z_x,Z_y,Z_y)$. Il punto tridimensionale da proiettare possiede le coordinate $Q(Q_x,Q_y,Q_z)$. Dall'equazione di una retta in forma vettoriale:

$$\vec{P} = \vec{P}_0 + s*a$$

potremmo ottenere quanto segue: Z si trova sulla retta e viene utilizzato per $\vec{P}_0$. A questo punto manca solo il vettore a che otterremo tramite il collegamento dei punti Z e Q:

$$\vec{a} = Q\text{-}Z = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \end{pmatrix} - \begin{pmatrix} Z_x \\ Z_y \\ Z_z \end{pmatrix} = \begin{pmatrix} Q_x\text{-}Z_x \\ Q_y\text{-}Z_y \\ Q_z\text{-}Z_z \end{pmatrix}$$

Con ciò otterremo la seguente equazione di un raggio (di una retta) dal punto dell'oggetto al centro:

$$P = Z + s* \begin{pmatrix} Q_x\text{-}Z_x \\ Q_y\text{-}Z_y \\ Q_z\text{-}Z_z \end{pmatrix}$$

oppure, sotto forma di parametri

$$P_x = Z_x + s*(Q_x\text{-}Z_x)$$
$$P_y = Z_y + s*(Q_y\text{-}Z_y)$$
$$P_z = Z_z + s*(Q_z\text{-}Z_z)$$

Come per la proiezione parallela, è il punto $\vec{P}$ ad essere oggetto della ricerca sul piano di proiezione, cioè il punto nel quale la retta interseca il piano (la lastra di vetro). A questo punto ci interessa sapere quale valore utilizzare per s, in modo che $\vec{P}$ si trovi veramente sul piano di proiezione. Anche in questo caso sarà facile se impostiamo la lastra di vetro al piano xy. Ciò, per il punto $\vec{P}$, significa che la coordinata Pz è uguale a 0. Quindi, dalla terza equazione, otterremo:

$$0 = Z_z + s* (Q_z\text{-}Z_z) \quad <=>$$

$$s = -\frac{Z_z}{Q_z\text{-}Z_z}$$

Trasportiamo ciò nelle prime due equazioni e trasformiamole leggermente:

$$P_x = Z_x - Z_z * \frac{Q_x - Z_x}{Q_z - Z_z} \quad \text{e}$$

$$P_y = Z_y - Z_z * \frac{Q_y - Z_y}{Q_z - Z_z}$$

Risparmiamoci per questa volta la lunga procedura di trasformazione e vediamo immediatamente il risultato:

$$P_x = \frac{Z_x*Q_z - Q_x*Z_z}{Q_z - Z_z} \quad \text{e}$$

$$P_y = \frac{Z_y*Q_z - Q_y*Z_z}{Q_z - Z_z}$$

dove:

P_x, P_y — coordinate del punto proiettato sul piano
Q_x, Q_y, Q_z — coordinate del punto spaziale da proiettare
Z_x, Z_y, Z_z — centro della proiezione

Affrontiamo ora la matrice di trasformazione corrispondente. Nella realtà essa non è molto semplice e per essa abbiamo bisogno in ogni caso delle coordinate omogenee, che fortunatamente avevamo introdotto in precedenza. Eccola:

$$\begin{aligned} \vec{P} &= (P_x*P_n \; P_y*P_n \; P_z*P_n \; P_n) \\ &= (Q_x*Q_n \; Q_y*Q_n \; Q_z*Q_n \; Q_n) * \begin{pmatrix} -Z_z & 0 & 0 & 0 \\ 0 & -Z_z & 0 & 0 \\ Z_x & Z_y & 0 & 1 \\ 0 & 0 & 0 & -Z_z \end{pmatrix} \end{aligned}$$

Tale matrice, abbreviata, sarà: $ZP(Z_x,Z_y,Z_z)$. I valori P_n e Q_n sono i fattori "omogenei" che, come noto, devono venire aggiunti con le coordinate omogenee (al momento dell'introduzione delle coordinate omogenee li abbiamo chiamati "n"). Normalmente essi vengono impostati a 1 (nei casi seguenti, anche Q_n potrebbe venire impostato uguale a 1, ma vogliamo determinare una formulazione generale, e allora Q_n potrebbe provenire da un'altra trasformazione; di conseguenza non sarebbe più uguale a 1). Tuttavia, in questo caso, essi hanno un ruolo ben preciso, anche se solo provvisoriamente. Infatti P_n non è uguale a 1. Cerchiamo di controllare brevemente se la matrice è esatta. La seguente verifica dovrebbe servire anche come esempio, nel momento in cui si volessere effettuare combinazioni di trasformazioni (per esempio proiezione centrale con rotazione ecc):

Il risultato dell'espressione precedente è:

$$\begin{aligned} \vec{P} &= (P_x*P_n \; P_y*P_n \; P_z*P_n \; P_n) \\ &= (-Q_x*Q_n*Z_z+Q_z*Q_n*Z_x \quad -Q_y*Q_n*Z_z+Q_z*Q_n*Z_y \quad 0 \quad Q_z*Q_n-Q_n*Z_zz) \end{aligned}$$

Se si hanno problemi con il calcolo di questa matrice, si potrà consultare l'appendice.

Al fine di calcolare i valori reali per P_x, P_y e P_z da questa matrice, dovremo dividere i singoli elementi della matrice per P_n (vedi definizione delle coordinate omogenee).

Dall'equazione per $\vec{P}$ possiamo ottenere i seguenti rapporti (si tratta semplicemente delle equazioni corrispondenti dei parametri):

$$P_x{*}P_n = -Q_x{*}Q_n{*}Z_z + Q_z{*}Q_n{*}Z_x$$
$$P_y{*}P_n = -Q_y{*}Q_n{*}Z_z + Q_z{*}Q_n{*}Z_y$$
$$P_z{*}P_n = 0$$
$$P_n = Q_z{*}Q_n - Q_n{*}Z_z$$

Trasformiamola come segue. A titolo di esempio, trasformiamo la prima tramite l'ultima:

$$P_x{*}P_n = -Q_x{*}Q_n{*}Z_z + Q_z{*}Q_n{*}Z_x \quad <=>$$

$$P_x = \frac{-Q_x{*}Q_n{*}Z_z + Q_z{*}Q_n{*}Z_x}{P_n} \quad <=>$$

$$P_x = \frac{-Q_x{*}Q_n{*}Z_z + Q_z{*}Q_n{*}Z_x}{Q_z{*}Q_n - Q_n{*}Z_z} <=>$$

$$P_x = \frac{-Q_x{*}Z_z + Q_z{*}Z_x}{Q_z - Z_z}$$

Si tratta esattamente della formula che avevamo precedentemente ottenuto. La seconda equazione viene risolta esattamente allo stesso modo e fornirà ugualmente il risultato desiderato. La terza viene trasformata ancora più semplicemente:

$$P_z{*}P_n = 0 \quad <=>$$
$$P_z = 0$$

Vediamo quindi che il trattamento della proiezione centrale diventa molto più complicato della rappresentazione parallela. Ciò è motivato anche dal fatto che in questo caso abbiamo a che fare con una divisione. Una divisione è possibile solo con coordinate omogenee, grazie all'aiuto dell'elemento omogeneo n (oppure, in questo caso, Q_n o P_n).

Tralasciamo per il momento un programma di esempio, rimandandolo a quando saremo più esperti nelle trasformazioni nel mondo tridimensionale. Infatti, già al momento della proiezione parallela, abbiamo dovuto fare ricorso alle rotazioni, cosa che per il momento vogliamo evitare.

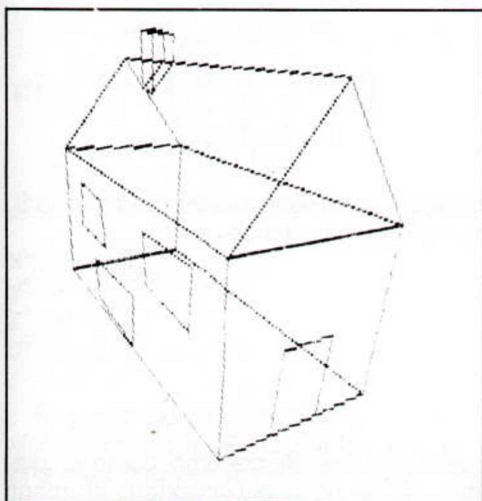


Fig. 4.21 Proiezione centrale 1

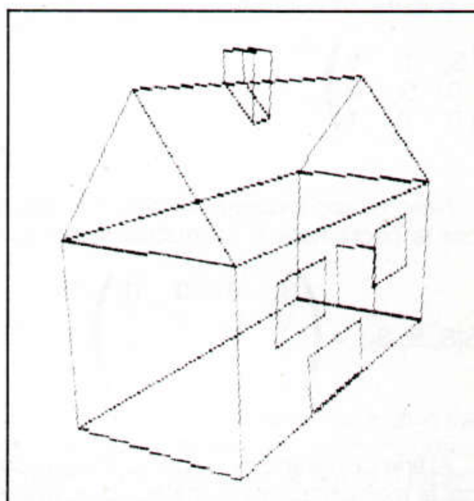


Fig. 4.22 Proiezione centrale 2

4.5 Come creare il movimento: trasformazioni anche nello spazio

In questi ultimi paragrafi abbiamo visto come è possibile proiettare immagini tridimensionali su di uno schermo bidimensionale. Nella realizzazione di programmi, tuttavia, abbiamo sentito la mancanza della possibilità di trasformazioni nello spazio (ricordiamoci della rotazione necessaria nel programma per la proiezione parallela).

D'altra parte abbiamo approfondito debitamente le trasformazioni sul piano e vorremmo restare su questa base. Cerchiamo quindi di approfondirla.

4.5.1 Spostamenti, ingrandimenti, rotazioni

All'inizio del capitolo ''Proiezioni'' abbiamo già visto come trasformare le matrici di trasformazione bidimensionali in forma di coordinate tridimensionali. Cerchiamo ora di ampliare tali matrici, in modo che funzionino anche nello spazio. Cominciamo quindi dalla trasformazione più semplice, la messa in scala (ingrandimento oppure riduzione).

La matrice bidimensionale per questa operazione era:

$$\begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

oppure, in coordinate omogenee:

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Nello spazio tridimensionale deve venire incorporato un fattore di ingrandimento S_z per la coordinata z. La matrice relativa, in coordinate omogenee, è:

$$S(S_x,S_y,S_z) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Al fine di ingrandire un punto P(x,y,z) con questa matrice, utilizzeremo, come sul piano, la moltiplicazione di matrici, che rappresenta uno dei nostri più importanti strumenti di lavoro (per la definizione della moltiplicazione fra matrici, ved. appendice):

$$\begin{aligned} P' &= (x'*n' \; y'*n' \; z'*n' \; n') \\ &= P(x,y,z) * S(S_x,S_y,S_z) \\ &= (x*n \; y*n \; z*n \; n) * \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= (x*S_x*n \; y*S_y*n \; z*S_z*n) \end{aligned}$$

e dal momento che n può venire impostato uguale a 1:

$$= (x*S_x \; y*S_y \; z*S_z \; 1)$$

Anche le forme parametriche risultano più semplici:

$$\begin{aligned} x' &= S_x*x \\ y' &= S_y*y \\ z' &= S_z*z \\ (n' &= n) \end{aligned}$$

Per la traslazione (spostamento) abbiamo bisogno in ogni caso delle coordinate omogenee. La matrice del piano era:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{pmatrix}$$

Nello spazio, l'amplieremo come segue:

$$T(T_x,T_y,T_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Le equazioni parametriche corrispondenti saranno quindi:

$$x' = x+T_x$$
$$y' = y+T_y$$
$$z' = z+T_z$$
$$(n' = n)$$

A questo punto ci mancano solo le rotazioni (se prescindiamo dalle molte altre trasformazioni possibili). Qui la faccenda si complica. Dal punto di vista bidimensionale, una rotazione veniva definita univocamente come rotazione attorno al punto 0 (oppure, più in seguito, attorno ad un altro punto a piacere). Nello spazio, l'enunciato "rotazione attorno ad un punto" non è più sufficiente. Infatti esistono infinite traiettorie, lungo le quali sarà possibile effettuare la rotazione. Nello spazio dovremo quindi indicare un asse attorno al quale vogliamo far ruotare l'oggetto. Solo a questo punto, le direzioni di rotazione si riducono a due: in senso orario o in senso antiorario.

Naturalmente possiamo pensare a un asse a piacere attorno al quale un'immagine può venire ruotata. Per semplicità, tuttavia, limitiamoci ai tre assi delle coordinate. Vedremo in seguito che con tali basi saranno possibili anche rotazioni attorno ad un asse a piacere.

Effettuiamo quindi il trasferimento delle formule dal bidimensionale al tridimensionale. Ipotizziamo che il piano bidimensionale si trovi nel sistema di coordinate tridimensionali esattamente sul piano xy (la coordinata z per tutti i punti del piano è uguale a 0).

Una rotazione del piano bidimensionale, trasportata nello spazio, sarebbe quindi una rotazione attorno all'asse z. Dal momento che la matrice di rotazione bidimensionale antioraria era la seguente in coordinate omogenee (a fornisce l'angolo di rotazione):

$$\begin{pmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

potremo esprimere la rotazione spaziale attorno all'asse z molto semplicemente tramite la seguente matrice:

$$R_z(a) = \begin{pmatrix} \cos(a) & \sin(a) & 0 & 0 \\ -\sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

oppure, sotto forma di parametri per la rotazione del punto P(x,y,z):

x' = x*cos(a) - y*sin(a)
y' = x*sin(a) + y*cos(a)
z' = z
(n' = n)

Come vediamo, la coordinata z, logicamente, non viene modificata.

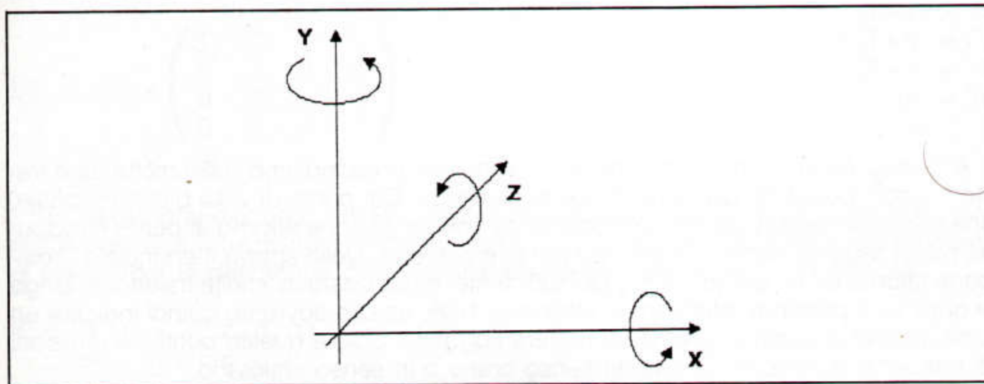


Fig. 4.23 Rotazioni attorno agli assi nello spazio

Resta ora la descrizione delle rotazioni attorno agli assi x ed y. Tutte le rotazioni devono avere luogo in senso antiorario. Per fare ciò, dovremo guardare dal lato positivo verso la punta di un asse (vedi figura 4.23). Le matrici di rotazione saranno quindi analoghe alla prima, per la rotazione attorno all'asse x:

$$R_x(a) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & \sin(a) & 0 \\ 0 & -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

con la forma parametrica per la rotazione di un punto P(x,y,z):

x' = x
y' = y*cos(a) - z*sin(a)
z' = y*sin(a) + z*cos(a)

e attorno all'asse y:

$$R_y(a) = \begin{pmatrix} \cos(a) & 0 & -\sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

con la forma parametrica per la rotazione di un punto P(x,y,z):

x' = x*cos(a) + z*sin(a)
y' = y
z' = -x*sin(a) + z*cos(a)

La matrice di trasformazione per una rotazione attorno a tutti gli assi con gli angoli a (asse x), b (asse y), e c (asse z) con la sequenza di rotazione attorno a x, attorno a y e attorno a z potrà venire calcolata tramite la seguente moltiplicazione tra matrici:

$R_x(a)_y(b)_z(c) = R_x(a) * R_y(b) * R_z(c)$

Abbiamo calcolato il risultato, che è il seguente:

$$R_x(a)_y(b)_z(c) = \begin{pmatrix} \cos(b)\cos(c) & \cos(b)\sin(c) & -\sin(b) & 0 \\ \sin(a)\sin(b)\cos(c) - \cos(a)\sin(c) & \sin(a)\sin(b)\sin(c) + \cos(a)\cos(c) & \sin(a)\cos(b) & 0 \\ \cos(a)\sin(b)\cos(c) + \sin(a)\sin(c) & \cos(a)\sin(b)\sin(c) - \sin(a)\cos(c) & \cos(a)\cos(b) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Se moltiplichiamo questa matrice gigantesca con quella di un punto, tale punto verrà ruotato in tutte e tre le dimensioni. Le equazioni di parametri risultanti saranno:

x' = x*A + y*B + z*C
y' = x*D + y*E + z*F
z' = x*G + y*H + z*I

Per i parametri A, B, ... dovremo immettere quanto segue:

A = cos(b) * cos(c)
B = cos(b) * sin(c)
C = -sin(b)
D = sin(a) * sin(b) * cos(c) - cos(a) * sin(c)
E = sin(a) * sin(b) * sin(c) + cos(a) * cos(c)
F = sin(a) * cos(b)
G = cos(a) * sin(b) * cos(c) + sin(a) * sin(c)
H = cos(a) * sin(b) * sin(c) - sin(a) * cos(c)
I = cos(a) * cos(b)

Nel caso in cui si voglia modificare la sequenza delle rotazioni (es., dapprima attorno all'asse z) si dovrà calcolare nuovamente la matrice. Con sequenze uguali potremo invece naturalmente lasciare uguali a zero uno o due angoli. A tal punto l'oggetto non verrà più ruotato attorno all'asse. Con ciò infatti alcuni seni e coseni delle equazioni precedenti diverrebbero sin(0) = 0 e cos(0) = 1.

Volendo proiettare su di un piano i punti ruotati, per poterli rappresentare sul video, dopo la rotazione dovranno venire sottoposti alla matrice di proiezione corrispondente. Ciò avrà luogo separatamente.

Vediamo ora con il seguente programma in C di trasformare in pratica ciò che fin'ora ci è parsa grigia teoria. Esso utilizza la maggior parte delle trasformazioni fin'ora sviluppate, compresa la proiezione centrale. A tale scopo esso impiega la struttura dati di un sistema tridimensionale CAD precedentemente citata, con la suddivisione in mondi, oggetti, linee, punti ecc. Ciascuna di queste unità viene rappresentata del programma con una o più strutture.

```
/*************************************/
/**                                **/
/**       Grande Demo per la       **/
/**                                **/
/**   ANIMAZIONE TRIDIMENSIONALE   **/
/**                                **/
/**              con               **/
/**  Proiez. centrale, Rotazione   **/
/**      Traslazione, Scala        **/
/**                                **/
/**         Organizzazione:        **/
/**   modello in filo metallico    **/
/**     orientato all'oggetto      **/
/**                                **/
/**           Axel Plenge          **/
/**                                **/
/*************************************/


#include <exec/types.h>
#include <intuition/intuition.h>
#include <libraries/mathffp.h>


/* Dichiarazioni delle funzioni (solo per Compilatore Aztec): */
/* a scelta anche: #include <functions.h>                     */
/*************************************************************/
VOID                       ClearScreen();
VOID                       Draw();
VOID                       Exit();


struct Message *           GetMsg();
VOID                       Move();
struct Library *           OpenLibrary();
struct Screen *            OpenScreen();
struct Window *            OpenWindow();
VOID                       ReplyMsg();
VOID                       ScreenToFront();
VOID                       SetAPen();
VOID                       SetRGB4();
LONG                       Wait();
/************************************************************/
```

```
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
LONG   *MathBase;


/* Struttura per l'inizializzazione di due nuovi schermi: */
/*********************************************************/

struct NewScreen NeuerBildschirm =
{
   0,                      /* Coordinata x superiore sinistra   */
                           /* Angolo (sempre 0)                 */
   0,                      /* Coordinata y superiore sinistra   */
                           /* Angolo                            */
   640,                    /* Larghezza schermo                 */
   256,                    /* Altezza schermo                   */
   2,                      /* Numero piani di immagine          */
   0,                      /* Colore dei dettagli               */
   1,                      /* Colore delle superfici            */
   HIRES,                  /* Modo grafico: 640x256             */
   CUSTOMSCREEN,           /* Tipo schermo                      */
   NULL;                   /* Nessun nuovo Font                 */
   "Proiezione centrale",/* Testo di intestazione schermo     */
   NULL,                   /* Inutilizzato, sempre ZERO (NULL)  */
   NULL,                   /* Nessun  BitMap proprio            */
};

/* Struttura per l'inizializzazione di due nuove finestre */
/*********************************************************/

struct NewWindow NeuesFenster =
{
   0,                      /* Coordinata x angolo super. sin.   */
   10,                     /* Coordinata y angolo super. sin.   */
   640,                    /* Larghezza finestra                */
   246,                    /* Altezza finestra                  */
   0,                      /* Colore dei dettagli               */
   1,                      /* Colore delle superfici            */
   MOUSEBUTTONS |          /* Risposta a pressione tasto Mouse  */
    RAWKEY,                /* e tasto                           */
   ACTIVATE |              /* Selezione elementi e tipo         */
    BORDERLESS|            /* della finestra: senza bordi       */
    NOCAREREFRESH |        /* nessuna segnalazione di Refresh   */
    RMBTRAP,               /* Nessuna operazione da menu        */
   NULL,                   /* nessun gadget proprio             */
   NULL,                   /* CheckMark                         */
   NULL,                   /* Testo di intestazione finestra    */
   0,                      /* Indirizzo struttura schermo       */
                           /* deve venire inizializzato entro   */
                           /* il programma dopo                 */
                           /* l'apertura di uno                 */
                           /* schermo                           */
   NULL,                   /* nessuna finestra SuperBitmap      */
   640,                    /* Larghezza minima                  */
   246,                    /* Altezza minima                    */
   640,                    /* Larghezza massima                 */
   256,                    /* Altezza massima                   */
   CUSTOMSCREEN,           /* Tipo schermo                      */
};
```

```
struct Screen *Bildschirm[2];
struct Window *Fenster[2];
struct RastPort *RastPort[2];
struct IntuiMessage *Message;


/* Colori tavolozza: */
LONG palette[4][3] =
   {
      { 0, 0, 0}, {15, 0, 1},
      { 0, 4,15}, {14,10, 1}
   };

VOID main()
{
LONG do_program();

LONG fehler;          /* Per le abbreviazioni dei tipi, ved. */
                      /* i File Exec-Include-File: types.h   */
ULONG i;


/* Apertura delle library di Intuition, Grafiche e Mathe    */

printf("3-D-Proiezione Centrale / C-Demo\n");

IntuitionBase =
(struct IntuitionBase *)OpenLibrary("intuition.library",0L);
if (IntuitionBase == NULL)            fehler = 1;
else
{
 GfxBase =
 (struct GfxBase *)OpenLibrary("graphics.library",0L);
 if (GfxBase == NULL)                 fehler = 2;
 else
 {
  MathBase =
  (LONG *)OpenLibrary("mathffp.library",0L);
  if (MathBase == NULL)               fehler = 3;
  else
  {
    /* Apertura schermo 0:              */
    Bildschirm[0] =
    (struct Screen *)OpenScreen(&NeuerBildschirm);
    if (Bildschirm[0] == NULL)         fehler = 4;
    else
    {
     /* Impostazione tavolozza colori: */
     for (i=0; i<4; i++)
      SetRGB4(&Bildschirm[0]->ViewPort, i,
              palette[i][0], palette[i][1], palette[i][2]);

     /* Apertura finestra 0:           */
     /* Aggiungere indirizzo della Struttura di Screen: */
     NeuesFenster.Screen = Bildschirm[0];
     Fenster[0] =
     (struct Window *)OpenWindow(&NeuesFenster);
     if (Fenster[0] == NULL)           fehler = 5;
```

```
   else
   {
    RastPort[0] = Fenster[0]->RPort; /* RastPort merken */

    /* Apertura schermo 1:            */
    Bildschirm[1] =
    (struct Screen *)OpenScreen(&NeuerBildschirm);
    if (Bildschirm[1] == NULL)        fehler = 6;
    else
    {
     /* Impostazione tavol. colori:  */
     for (i=0; i<4; i++)
      SetRGB4(&Bildschirm[1]->ViewPort, i,
              palette[i][0], palette[i][1], palette[i][2]);

     /* Apertura finestra 1:          */
     /* Aggiungere indirizzo della Struttura di Screen: */
     NeuesFenster.Screen = Bildschirm[1];
     Fenster[1] =
     (struct Window *)OpenWindow(&NeuesFenster);
     if (Fenster[1] == NULL)          fehler = 7;
     else
     {
      RastPort[1] = Fenster[1]->RPort; /* RastPort merken */

      fehler = do_program();        /* Chiamata del pro-      */
                                    /* gramma vero e proprio */

      CloseWindow(Fenster[1]);      /* Chiusura finestra 1    */
     }
     CloseScreen(Bildschirm[1]);    /* Chiusura Screen 1      */
    }
    CloseWindow(Fenster[0]);        /* Chiusura finestra 0    */
   }
   CloseScreen(Bildschirm[0]);      /* Chiusura Screen 0      */
  }
  CloseLibrary(MathBase);           /* Mathe-Library close    */
 }
 CloseLibrary(GfxBase);             /* Graphics-Lib. close    */
}
CloseLibrary(IntuitionBase);        /* Intuition-Lib. close   */
}
Exit(fehler);                       /* Uscita con cod.errore */
}
```

```
/************************/
/* Programma principale */
/************************/

/* Variabili globali e strutture: */
/*********************************/

/* Addizione di 90 ad un angolo w con: 0 <= w < 360        */
/* anche l'angolo risultante si trovera' fra  0 e 360      */
/* (Per il calcolo del coseno si e' utilizzata la          */
/* seguente Tabella seno)                                  */


#define W_PLUS_90(w)    ( ((w) >= 270)? (w)-270 : (w)+90 )


/* Tabella Seno/Coseno:                       */

/* La tabella contiene, a distanze di 1 Grado */
/* i valori di seno da 0 a 359 gradi, sempre  */
/* moltiplicati per 2^15=32768.               */
/* Il bit piu' alto e' quello del segno.      */
/* Per la determinazione del Coseno si        */
/* dovranno aggiungere sempre, in precedenza, */
/* 90 gradi all'angolo.                       */

WORD sinus[] =
 {
  0x0000, 0x023C, 0x0478, 0x06B3, 0x08EE, /*  0- 4 Gradi */
  0x0B28, 0x0D61, 0x0F99, 0x11D0, 0x1406, /*  5- 9 Gradi */
  0x163A, 0x186C, 0x1A9D, 0x1CCB, 0x1EF7, /* 10-14 Gradi */
  0x2121, 0x2348, 0x256C, 0x278E, 0x29AC, /* ...         */
  0x2BC7, 0x2DDF, 0x2FF3, 0x3203, 0x3410,
  0x3618, 0x381D, 0x3A1C, 0x3C18, 0x3E0E,
  0x4000, 0x41ED, 0x43D4, 0x45B7, 0x4794,
  0x496B, 0x4B3D, 0x4D08, 0x4ECE, 0x508E,
  0x5247, 0x53FA, 0x55A6, 0x574C, 0x58EB,
  0x5A82, 0x5C13, 0x5D9D, 0x5F1F, 0x609A,
  0x620E, 0x637A, 0x64DE, 0x663A, 0x678E,
  0x68DA, 0x6A1E, 0x6B5A, 0x6C8D, 0x6DB8,
  0x6EDA, 0x6FF4, 0x7104, 0x720D, 0x730C,
  0x7402, 0x74EF, 0x75D3, 0x76AE, 0x7780,
  0x7848, 0x7907, 0x79BC, 0x7A68, 0x7B0B,
  0x7BA3, 0x7C33, 0x7CB8, 0x7D34, 0x7DA6,
  0x7E0E, 0x7E6D, 0x7EC1, 0x7F0C, 0x7F4C,
  0x7F83, 0x7FB0, 0x7FD3, 0x7FEC, 0x7FFB,

  0x7FFF, 0x7FFB, 0x7FEC, 0x7FD3, 0x7FB0, /* 90-94 Gradi */
  0x7F83, 0x7F4C, 0x7F0C, 0x7EC1, 0x7E6D, /* 95-99 Gradi */
  0x7E0E, 0x7DA6, 0x7D34, 0x7CB8, 0x7C33, /* ...         */
  0x7BA3, 0x7B0B, 0x7A68, 0x79BC, 0x7907,
  0x7848, 0x7780, 0x76AE, 0x75D3, 0x74EF,
  0x7402, 0x730C, 0x720D, 0x7104, 0x6FF4,
  0x6EDA, 0x6DB8, 0x6C8D, 0x6B5A, 0x6A1E,
  0x68DA, 0x678E, 0x663A, 0x64DE, 0x637A,
```

```
  0x620E, 0x609A, 0x5F1F, 0x5D9D, 0x5C13,
  0x5A82, 0x58EB, 0x574C, 0x55A6, 0x53FA,
  0x5247, 0x508E, 0x4ECE, 0x4D08, 0x4B3D,
  0x496B, 0x4794, 0x45B7, 0x43D4, 0x41ED,
  0x4000, 0x3E0E, 0x3C18, 0x3A1C, 0x381D,
  0x3618, 0x3410, 0x3203, 0x2FF3, 0x2DDF,
  0x2BC7, 0x29AC, 0x278E, 0x256C, 0x2348,
  0x2121, 0x1EF7, 0x1CCB, 0x1A9D, 0x186C,
  0x163A, 0x1406, 0x11D0, 0x0F99, 0x0D61,
  0x0B28, 0x08EE, 0x06B3, 0x0478, 0x023C,

  0x0000, 0xFDC4, 0xFB88, 0xF94D, 0xF712, /* 180-184 Gradi */
  0xF4D8, 0xF29F, 0xF067, 0xEE30, 0xEBFA, /* ...            */
  0xE9C6, 0xE794, 0xE563, 0xE335, 0xE109,
  0xDEDF, 0xDCB8, 0xDA94, 0xD872, 0xD654,
  0xD439, 0xD221, 0xD00D, 0xCDFD, 0xCBF0,
  0xC9E8, 0xC7E3, 0xC5E4, 0xC3E8, 0xC1F2,
  0xC000, 0xBE13, 0xBC2C, 0xBA49, 0xB86C,
  0xB695, 0xB4C3, 0xB2F8, 0xB132, 0xAF72,
  0xADB9, 0xAC06, 0xAA5A, 0xA8B4, 0xA715,
  0xA57E, 0xA3ED, 0xA263, 0xA0E1, 0x9F66,
  0x9DF2, 0x9C86, 0x9B22, 0x99C6, 0x9872,
  0x9726, 0x95E2, 0x94A6, 0x9373, 0x9248,
  0x9126, 0x900C, 0x8EFC, 0x8DF3, 0x8CF4,
  0x8BFE, 0x8B11, 0x8A2D, 0x8952, 0x8880,
  0x87B8, 0x86F9, 0x8644, 0x8598, 0x84F5,
  0x845D, 0x83CD, 0x8348, 0x82CC, 0x825A,
  0x81F2, 0x8193, 0x813F, 0x80F4, 0x80B4,
  0x807D, 0x8050, 0x802D, 0x8014, 0x8005,

  0x8000, 0x8005, 0x8014, 0x802D, 0x8050, /* 270-274 Gradi */
  0x807D, 0x80B4, 0x80F4, 0x813F, 0x8193, /* ...            */
  0x81F2, 0x825A, 0x82CC, 0x8348, 0x83CD,
  0x845D, 0x84F5, 0x8598, 0x8644, 0x86F9,
  0x87B8, 0x8880, 0x8952, 0x8A2D, 0x8B11,
  0x8BFE, 0x8CF4, 0x8DF3, 0x8EFC, 0x900C,

  0x9126, 0x9248, 0x9373, 0x94A6, 0x95E2,
  0x9726, 0x9872, 0x99C6, 0x9B22, 0x9C86,
  0x9DF2, 0x9F66, 0xA0E1, 0xA263, 0xA3ED,
  0xA57E, 0xA715, 0xA8B4, 0xAA5A, 0xAC06,
  0xADB9, 0xAF72, 0xB132, 0xB2F8, 0xB4C3,
  0xB695, 0xB86C, 0xBA49, 0xBC2C, 0xBE13,
  0xC000, 0xC1F2, 0xC3E8, 0xC5E4, 0xC7E3,
  0xC9E8, 0xCBF0, 0xCDFD, 0xD00D, 0xD221,
  0xD439, 0xD654, 0xD872, 0xDA94, 0xDCB8,
  0xDEDF, 0xE109, 0xE335, 0xE563, 0xE794,
  0xE9C6, 0xEBFA, 0xEE30, 0xF067, 0xF29F,
  0xF4D8, 0xF712, 0xF94D, 0xFB88, 0xFDC4  /* 355-359 Gradi */
 };

struct punkt        /* Struttura punto nello spazio        */
{
   WORD  x;                 /* Coordinata x del punto      */
   WORD  y;                 /* Coordinata y del punto      */
   WORD  z;                 /* Coordinata z del punto      */
};
```

```
struct linie        /* Definizione linea    */
{
   WORD  p1;                  /* Nr. del primo punto finale     */
   WORD  p2;                  /* Nr. del secondo punto finale   */
};

struct objekt       /* Struttura oggetto    */
{
   UWORD  anz_pun;            /* Numero dei punti               */
   struct punkt *punkte;      /* Puntatore alla matrice punti   */
   UWORD  anz_lin;            /* Numero delle linee             */
   struct linie *linien;      /* Puntatore alla matrice linee   */
   UWORD  farbe;              /* Colore oggetto                 */
   char   *name;              /* Nome oggetto                   */
   UBYTE  sichtbar;           /* =0: invisibile                 */
                              /* =1: visibile                   */
   UBYTE  transform;          /* =0: l'oggetto non viene        */
                              /*     trasformato                */
                              /* =1: l'oggetto viene            */
                              /*     trasformato                */
};

struct welt         /* Tutto cio' che si trova nel mondo */
{
   UWORD  anz_obj;            /* Numero degli oggetti           */
   struct objekt *objekte;    /* Puntatore alla matrice ogget.  */

   /* Valori di trasformazione per l'intero mondo */

   WORD   x_rot;              /* Angolo di rotazione (Gradi)    */
   WORD   y_rot;
   WORD   z_rot;
   WORD   x_ska;              /* Valori di scala in             */
   WORD   y_ska;              /* decimi                         */
   WORD   z_ska;
   WORD   x_tra;              /* Valori di traslazione          */
   WORD   y_tra;
   WORD   z_tra;
   WORD   x_beo;              /* Coordinate dell'               */
   WORD   y_beo;              /* osservatore                    */
   WORD   z_beo;

   WORD   xe_tra;             /* Traslazione sul piano          */
   WORD   ye_tra;             /* dopo la proiezione             */
   WORD   xe_ska;             /* e scala del piano per adegua-  */
   WORD   ye_ska;             /* mento alla risoluzione         */
                              /* dello schermo                  */

} urwelt =
   { 2,0,
     340,20,0, 20,20,20, 0,0,0, 0,0,-500,
     200,100, 2,1 };

USHORT Fen_Nr_s = 1,          /* Nr. finestra visibile          */
       Fen_akt  = 1,          /* Finestra attiva                */
       Fen_Nr_v = 0;          /* Nr. finestra nascosta          */
```

```
/* Valori per incremento/diminuzione dei */
/* Valori di trasformazione da tastiera: */
#define SKALIER_INC      1  /* Valore Inc per scala           */
#define DREH_INC         1  /* Valore Inc per rotazione       */
#define TRANSLA_INC      4  /* Valore Inc per traslazione z   */
#define BEOB_INC        15  /* + coordinata z osservatore     */

/* Definizione codici tasti RAW: */
#define SHIFT       (IEQUALIFIER_LSHIFT | IEQUALIFIER_RSHIFT)
#define PLUS         0x5e
#define PLUS2        0x1b
#define MINUS        0x4a
#define MAL          0x5d
#define GETEILT      0x5c
#define C_RECHTS     0x4e
#define C_LINKS      0x4f
#define C_AUF        0x4c
#define C_AB         0x4d
#define Z_C_AUF      0x3e
#define Z_C_AB       0x1e
#define Z_C_RECHTS   0x2f
#define Z_C_LINKS    0x2d
#define DEL          0x46
#define HELP         0x5f
#define Z_ENTER      0x43
#define Z_NULL       0x0f
#define Z_PUNKT      0x3c
#define ESC          0x45

LONG do_program()

{
   VOID  welt_init(),
         schaffe_welt(),
         init_ausgabe(),
         welt_verteiler(),
          pun_ska(),
          pun_tra(),
          pun_rot(),
          pun_zpj(),
          obj_zeichnen(),
         add_winkel();

   struct welt   welt;           /* Memoria per struttura mondo            */
   struct objekt *objekte;       /* Matrice di tutti gli oggetti origin.   */
   struct objekt *trans_obj;     /* Matrice degli oggetti trasformati      */

   ULONG  class;                 /* Memoria per la struttura               */
   USHORT code, qualifier;       /* di messaggio di Intuition              */
   APTR   address;
   SHORT  mouse_x, mouse_y;

   WORD x_rot_inc = 0,           /* Incrementi di rotazione attuali        */
        y_rot_inc = 0,
        z_rot_inc = 0;
```

```
USHORT Fen_zwis;           /* Double-Buff memorizzazione intermedia */
USHORT verd_flag = 1;      /* Flag per tracciatura                  */
                           /* nascosta/non nascosta                 */

if (!verd_flag)
{
   Fen_zwis = Fen_Nr_v;   /* Per tracciatura non nascosta          */
   Fen_Nr_v = Fen_Nr_s;
}

/* Inizializzazione struttura mondo: */
welt_init(&welt);

/* Caricamento dati per mondo:       */
schaffe_welt(&welt, &objekte, &trans_obj);


/* Loop di disegno:                  */
/*************************************/
code = 0;
while (code != ESC)        /* Loop in funzione fino a ESC */
{
   REGISTER UWORD flag = 0;

   /* Trasferimento dati oggetto in matrici    */
   /* per oggetti trasformati                  */
   init_ausgabe(&welt, trans_obj);

   /* Trasformazione di tutto il mondo: */
   welt_verteiler(1, &welt, trans_obj); /* Scala          */
   welt_verteiler(2, &welt, trans_obj); /* Traslazione    */
   welt_verteiler(3, &welt, trans_obj); /* Rotazione      */

   /* Proiezione di tutto il mondo: */
   welt_verteiler(4, &welt, trans_obj);


   /* Disegno di tutto il mondo, eventualmente nascosto  */

   Move(RastPort[Fen_Nr_v], 0L, 0L);     /* Finestra       */
   ClearScreen(RastPort[Fen_Nr_v]);      /* cancellazione */

   welt_verteiler(5, &welt, trans_obj); /* disegno        */

   if (verd_flag)
   {
      /* Apparizione davanti della finestra nascosta: */
      /* und Fen_Nr_v <-> Fen_Nr_s tauschen:    */

      ScreenToFront( Bildschirm[Fen_Nr_v] );

      Fen_zwis = Fen_Nr_v;
      Fen_Nr_v = Fen_Nr_s;
      Fen_Nr_s = Fen_zwis;
   }
```

```
/* Loop di Input: */
/*****************/
do
{
   /* Controllare tasto e eventualmente leggere nel codice */
   /* Attendere disegno solo se non ci sono rotazioni      */
   /* Viene letto il codice tasto (non l' ASCII!)          */

   if (x_rot_inc == 0  &&
       y_rot_inc == 0  &&
       z_rot_inc == 0)
   {
      /* Attesa messaggio: */
      Wait(1L << Fenster[Fen_akt]->UserPort->mp_SigBit);
      flag = 1;              /* Flag per attesa      */
   }

   /* Elaborazione segnalazione: */
   /*****************************/
   while (Message =
          (struct IntuiMessage *)GetMsg(Fenster[Fen_akt]->UserPort))
   {
      /* Daten aus Message-Struktur lesen: */
      class     = Message->Class;
      code      = Message->Code;
      qualifier = Message->Qualifier;
      address   = Message->IAddress;
      mouse_x   = Message->MouseX;
      mouse_y   = Message->MouseY;
      ReplyMsg(Message);    /* Restituzione messaggio  */

      /* In caso di tasto Mouse (solo se e' premuto     */
      /* il tasto sinistro del Mouse)                   */
      if (class == MOUSEBUTTONS && code == MENUDOWN)
      {
         /* posizionamento del mondo sullo schermo:    */
         welt.xe_tra = mouse_x;
         welt.ye_tra = mouse_y;

         flag = 0;
      }
      else     /* diversamente: tastiera     */
      {
         switch (code)
         {
            case PLUS2:
            case PLUS:                 /* Ingrandimento   */
               welt.x_ska += SKALIER_INC,
               welt.y_ska += SKALIER_INC,
               welt.z_ska += SKALIER_INC;
               flag = 0;
               break;
            case MINUS:                /* Riduzione       */
               welt.x_ska -= SKALIER_INC,
               welt.y_ska -= SKALIER_INC,
               welt.z_ska -= SKALIER_INC;
               flag = 0;
               break;
```

```
case C_RECHTS:
   if (SHIFT & qualifier) /* SHIFT?        */
   {
      /* si: Rotazione a destra attorno all'asse z */
      z_rot_inc -= DREH_INC;
   }
   else
   {
      /* no: Rotazione a destra attorno all'asse y  */
      y_rot_inc -= DREH_INC;
   }
   flag = 0;
   break;
case C_LINKS:
   if (SHIFT & qualifier) /* SHIFT?        */
   {

   /* Si:  Rotazione a sinistra attorno asse z    */
   z_rot_inc += DREH_INC;

   }
   else
   {
      /* no: Rotazione a sinistra attorno asse y      */
      y_rot_inc += DREH_INC;
   }
   flag = 0;
   break;
case C_AUF: /* Rotazione a destra attorno asse x      */
   x_rot_inc -= DREH_INC;
   flag = 0;
   break;
case C_AB:  /* Rotazione a sinistra attorno asse x    */
   x_rot_inc += DREH_INC;
   flag = 0;
   break;
case Z_C_RECHTS:  /* Allontanamento mondo              */
   welt.z_tra += TRANSLA_INC;
   flag = 0;
   break;
case Z_C_LINKS:    /* Avvicinamento mondo              */
   welt.z_tra -= TRANSLA_INC;
   flag = 0;
   break;
case Z_C_AUF:       /* Allontanamento osservatore      */
   welt.z_beo -= BEOB_INC;
   flag = 0;
   break;
case Z_C_AB:        /* Avvicinamento osservatore       */
   welt.z_beo += BEOB_INC;
   flag = 0;
   break;
case DEL:      /* Inserzione si/no sistema coordinate */
   ((welt.objekte)+0)->sichtbar =
     (((welt.objekte)+0)->sichtbar)?  0 : 1;
   flag = 0;
   break;
```

```
            case HELP:    /* Arrestare tutto e ritorno                    */
               /* Ripristino dei valori del mondo                         */
               welt_init(&welt);
            case Z_ENTER:/* Arresto solo delle rotazioni                  */
               x_rot_inc =     /* Arresto rotazioni                       */
               y_rot_inc =
               z_rot_inc = 0;
               flag = 0;
               break;
            case Z_NULL: /* Trasformazione si/no del sistema              */
                         /* delle coordinate                              */
               ((welt.objekte)+0)->transform =
                  (((welt.objekte)+0)->transform)?  0 : 1;
               flag = 0;
               break;
            case Z_PUNKT:/* Disegno nascosto si/no      */
               if (verd_flag = (verd_flag)? 0 : 1)
               {
                  Fen_Nr_v = Fen_zwis; /* si            */
               }
               else
               {
                  Fen_zwis = Fen_Nr_v; /* no            */
                  Fen_Nr_v = Fen_Nr_s;
               }
               break;
            case ESC:                   /* Fine          */
               flag = 0;
               break;

               } /* switch */
             } /* else */
          } /* while */
          add_winkel(&welt.x_rot, x_rot_inc);
          add_winkel(&welt.y_rot, y_rot_inc);
          add_winkel(&welt.z_rot, z_rot_inc);

      } while (flag);            /* finche' flag != 0          */
   } /* while */
   return((LONG)TRUE);
}

/* Aggiunta velore pos./neg. ad un angolo: */
/*******************************************/

VOID add_winkel(winkel, inc)
WORD *winkel;
WORD inc;

{
*winkel += inc;              /* Aggiunta valori       */
if (*winkel >= 360)          /* portando l'angolo a */
   *winkel -= 360;           /* valori da 0 a 359     */
if (*winkel < 0)
   *winkel += 360;
}
```

```
/* Inizializzazione struttura mondo: */
/************************************/

VOID welt_init(w)
struct welt *w;

{
   urwelt.anz_obj = w->anz_obj; /* Numero invariato oggetti */
   urwelt.objekte = w->objekte; /* e puntatori oggetti       */

   *w = urwelt;                 /* Mondo origin. dopo mondo */
}


/****************************************/
/* Preparazione di tutte le strutt. dati */
/****************************************/

/* Questa matrice indica quanti punti e     */
/* quante linee sono possedute dai diversi */
/* oggetti, nonche' quali colori e nomi     */
/* essi abbiano                             */

struct new_objekt
{
   UWORD anz_pun;      /* Numero punti  */
   UWORD anz_lin;      /* Numero Linee  */
   UWORD farbe;        /* Colori        */
   char *name;         /* Nome          */
} new_objekt[] =
   {
      { 6, 3,                        /* Punti/Linee oggetto 1    */
        2, "Koordinatensystem"},     /* Colori/Nome oggetto 1    */

      {34,43,                        /* Punti/Linee oggetto 2    */
        3, "Haus"},                  /* Colori/Nome oggetto 2    */

      {0,0,0,"\0"}                   /* Simbolo di fine          */
   };


/* Questa matrice contiene tutti i punti di tutti */
/* gli oggetti. La sequenza degli oggetti e dei   */
/* punti deve venire sempre rispettata!           */

struct punkt _punkte[] =
   {
      /* Sistema di coordinate: */
      {-15,  0,  0}, { 40,  0,  0}, {  0,-15,  0},
      {  0, 40,  0}, {  0,  0,-15}, {  0,  0, 40},
```

```
        /* Casa:                        */
        {-6, 6, 14}, { 6, 6, 14}, { 6,-6, 14},
        {-6,-6, 14}, { 6, 6,-14}, {-6, 6,-14},
        {-6,-6,-14}, { 6,-6,-14}, { 0,14, 14},
        { 0,14,-14}, {-2,-6, 14}, {-2, 0, 14},
        { 2, 0, 14}, { 2,-6, 14}, { 6, 4, 10},
        { 6, 4,  4}, { 6, 0,  4}, { 6, 0, 10},
        { 6, 4, -4}, { 6, 4,-10}, { 6, 0,-10},
        { 6, 0, -4}, { 6,-2,  2}, { 6,-2, -6},
        { 6,-6, -6}, { 6,-6,  2}, { 2,12, -4},
        { 2,16, -4}, { 2,16, -6}, { 2,12, -6},
        { 0,14, -4}, { 0,16, -4}, { 0,16, -6},
        { 0,14, -6}
    };

/* Questa matrice contiene tutte le linee di tutti */
/* gli oggetti. Rispettare la sequenza!            */
/* La numerazione dei punti e' sempre relativa     */
/* al primo punto di un oggetto !                  */
/* (Primo punto dell'oggetto uguale a zero)        */

struct linie _linien[] =
    {
        /* Sistema di coordinate: */
        {0,1}, {2,3}, {4,5},

        /* Casa:                        */
        { 0, 1}, { 1, 2}, { 2, 3}, { 3, 0},
        { 1, 4}, { 4, 7}, { 7, 2}, { 7, 6},
        { 6, 5}, { 5, 4}, { 5, 0}, { 6, 3},
        { 8, 9}, { 0, 8}, { 8, 1}, { 4, 9},
        { 5, 9}, {10,11}, {11,12}, {12,13},
        {14,15}, {15,16}, {16,17}, {17,14},
        {18,19}, {19,20}, {20,21}, {21,18},
        {22,23}, {23,24}, {24,25}, {25,22},
        {26,27}, {27,28}, {28,29}, {29,26},
        {30,31}, {31,32}, {32,33}, {26,30},
        {29,33}, {27,31}, {28,32}
    };

/* Riserva memoria: */
/********************/

/* Numero oggetti: */
#define ANZ_OBJ (sizeof new_objekt / sizeof(struct new_objekt) - 1)

/* Numero Punti: */
#define ANZ_PUN (sizeof _punkte / sizeof(struct punkt))

struct objekt _objekte[ANZ_OBJ];     /* Riserv. per oggetti    */
struct objekt _trans_obj[ANZ_OBJ];   /* per oggetti trasform.  */
struct punkt _trans_pun[ANZ_PUN];    /* Riserva mem.per punti  */
                                     /* trasformati            */
```

```
/* Lettura di tutte le definizioni di */
/* mondo, oggetto, linea e punto      */
/************************************/

VOID schaffe_welt(w, p_objekte, p_trans_obj)
struct welt *w;                /* Puntat. alla strutt. mondo */
struct objekt **p_objekte;     /* Puntat. al puntatore alla  */
struct objekt **p_trans_obj;   /* matrice dell'oggetto       */

{
   REGISTER UWORD i;
   REGISTER UWORD p_ges = 0,   /* Numero totale attuale punti*/
                  l_ges = 0;   /* Numero totale attuale linee*/

   *p_objekte = _objekte;      /* Indirizzo matrice oggetto  */
   *p_trans_obj = _trans_obj; /* Indirizzo matrice oggetto  */
                               /* per matrice trasformata    */
   w->objekte = _objekte;      /* Indirizzo matrice oggetto  */
                               /* nella struttuta mondo      */

   for (i=0; new_objekt[i].anz_pun != 0; i++)
   {
      /* Occupazione struttura oggetto per ogni oggetto: */
      _objekte[i].anz_pun = new_objekt[i].anz_pun; /* Nr.punti */
      _objekte[i].punkte  = &_punkte[p_ges];       /* Matr.Pun */
      _objekte[i].anz_lin = new_objekt[i].anz_lin; /* Nr.linee */
      _objekte[i].linien  = &_linien[l_ges];       /* Matr.lin */
      _objekte[i].farbe   = new_objekt[i].farbe;   /* Colore   */
      _objekte[i].name    = new_objekt[i].name;    /* Nome     */
      _objekte[i].sichtbar= 1;                     /* visibile */
      _objekte[i].transform = 1;                   /*trasf.    */

      p_ges += new_objekt[i].anz_pun; /* n.ro tot.att.punti */
      l_ges += new_objekt[i].anz_lin; /* n.ro tot.att.Linee */
   }
   w->anz_obj = i;        /* Inserimento numero oggetti */
}


/* Preparazione compito e trasformazione tramite  */
/* trasferimento dei dati originali dell'oggetto  */
/* in matrici per i dati trasformati              */
/************************************************/

VOID init_ausgabe(w, trans_obj)
struct welt *w;                  /* Puntatore struttura mondo */
struct objekt *trans_obj;        /* Puntatore agli oggetti    */
                                 /* trasformati               */
```

```
{
   struct objekt *objekte;              /* Puntatore agli oggetti      */
   REGISTER UWORD i, k;                 /* Indici loop FOR             */
   REGISTER UWORD p_ges = 0;            /* n.ro totale attuale punti */

   objekte = w->objekte;                /* Inizializzaz. puntatore     */

   for (i=0; i < w->anz_obj; i++)      /* Elaboraz. di ogni oggetto */
   {
      trans_obj[i] = objekte[i];        /* Trasferimento struttura     */
                                        /* oggetto in una              */
                                        /* struttura per oggetti       */
                                        /* trasformati                 */

      /* Indirizzo dell'elemento di matrice corrispondente come    */
      /* indirizzo primo elemento matrice dei punti dell'oggetto */
      trans_obj[i].punkte = &_trans_pun[p_ges];

      p_ges += objekte[i].anz_pun;   /* Startpos. next Array */

      for (k=0; k < objekte[i].anz_pun; k++)
      {
         /* Trasferimento matrici punti */
         trans_obj[i].punkte[k] = objekte[i].punkte[k];
      }
   }
}


/* Elaborazione dell'intero mondo */
/**********************************/

VOID welt_verteiler(modus, w, trans_obj)
UWORD modus;                     /* Tipo di elaborazione*/
                                 /* =1: scala           */
                                 /* =2: Traslazione     */
                                 /* =3: Rotazione       */
                                 /* =4: Proiez. centr.  */
                                 /* =5: Disegno         */
struct welt *w;                  /* Puntatore a struttura mondo    */
struct objekt *trans_obj;        /* Puntatore matrici strutture    */
                                 /* per oggetti trasformati        */

{
   VOID  obj_transf(),
          pun_ska(),
          pun_tra(),
          pun_rot(),
          pun_zen(),
         obj_zeichnen();

   REGISTER UWORD i;

   /* Trasformazione di tutti gli oggetti */
   for (i=0; i < w->anz_obj; i++)
   {
      /* Effettuare tutte le elaborazione all'oggetto attuale */
      /* memorizzare eventuali eventi in trans_obj[]          */
      switch (modus)
```

```
      {
         case 1:
         /* if (trans_obj[i].transform) *//* trasformare? */
            {
               obj_transf(pun_ska, &trans_obj[i],
                          w->x_ska, w->y_ska, w->z_ska);
            }
            break;
         case 2:
            if (trans_obj[i].transform) /* trasformare? */
            {
            obj_transf(pun_tra, &trans_obj[i],
                       w->x_tra, w->y_tra, w->z_tra);
            }
            break;
         case 3:
            if (trans_obj[i].transform) /* trasformare? */
            {
            obj_transf(pun_rot, &trans_obj[i],
                       w->x_rot, w->y_rot, w->z_rot);
            }
            break;
         case 4:
            obj_transf(pun_zen, &trans_obj[i],
                       w->x_beo, w->y_beo, w->z_beo);
            break;
         case 5:
            obj_zeichnen(w, &trans_obj[i]);
            break;
      } /* switch */
   } /* for */


/* Trasformazione di un oggetto : */
/**********************************/

VOID obj_transf(operation, objekt, t1, t2, t3)
VOID (*operation)();      /* Puntatore alla funzione */
                          /* che deve eseguire la    */
                          /* trasformazione:         */
                          /* pun_ska(), pun_tra(),   */
                          /* pun_rot(), pun_zen()    */
struct objekt *objekt;    /* Indirizzo dell'oggetto  */
                          /* da trasformare          */
WORD t1, t2, t3;          /* Parametri di trasform.  */

{

   struct punkt *punkte;        /* Puntat. matrice di punti    */
   REGISTER UWORD i;

   punkte = objekt->punkte;     /* Indirizzo matrice di punti */

   /* Trasformazione di tutti i punti: */
   for (i=0; i < objekt->anz_pun; i++)
   {
```

```
        /* Chiamata indiretta della routine di trasformazione */
        (*operation)(&punkte[i], t1, t2, t3); /*1 Pkt transf.*/
    }
}


/* Messa in scala di un punto */
/*****************************/

VOID pun_ska(punkt, xs, ys, zs)
struct punkt *punkt;            /* Puntat. alla strutt. di punto */
WORD xs, ys, zs;                /* Parametri di messa in scala   */

{
   punkt->x = (punkt->x * xs) / 10;   /* Fattore di scala in     */
   punkt->y = (punkt->y * ys) / 10;   /* decimi                  */
   punkt->z = (punkt->z * zs) / 10;
}


/* Traslazione di un punto */
/**************************/

VOID pun_tra(punkt, xl, yl, zl)
struct punkt *punkt;
WORD xl, yl, zl;

{
   punkt->x += xl;
   punkt->y += yl;
   punkt->z += zl;
}


/* Rotazione di un punto */
/************************/

VOID pun_rot(punkt, xr_w, yr_w, zr_w)
struct punkt *punkt;
WORD xr_w, yr_w, zr_w;          /* Angolo di rotazione          */

{
   REGISTER LONG x,y,z;          /* Proposte di registro         */
   REGISTER LONG sin_w, cos_w;
   LONG zwis;

   x = punkt->x;         /* Caricamento registro con coordinate*/
   y = punkt->y;
   z = punkt->z;

   /* Rotazione attorno all'asse z: */
   if (xr_w)                    /* Angolo x   != 0 ?   */
   {
      /* Prelevare seno/coseno per 2^15 dalla tabella */
      sin_w = sinus[xr_w];
      cos_w = sinus[ W_PLUS_90(xr_w) ];
```

```
    /* Calcolo della matrice di rotazione e divisione per 2^15 */
    /* (Calcolo all'indietro dei valori di seno/coseno)        */
    zwis = (y*cos_w - z*sin_w) >> 15;
    z    = (y*sin_w + z*cos_w) >> 15;
    y    = zwis;
}

/* Rotazione attorno all'asse y */
if (yr_w)                  /* angolo y != 0 ?   */
{
    /* Prelevare seno/coseno per 2^15 dalla tabella */
    sin_w = sinus[yr_w];
    cos_w = sinus[ W_PLUS_90(yr_w) ];

    /* Calcolo della matrice di rotazione e divisione per 2^15 */
    /* (Calcolo all'indietro dei valori di seno/coseno)        */
    zwis = ( x*cos_w + z*sin_w) >> 15;
    z    = (-x*sin_w + z*cos_w) >> 15;
    x    = zwis;
}
   /* Rotazione attorno all'asse z */
   if (zr_w)                   /* Angolo z != 0 ?   */
   {
      /* Prelevare seno/coseno per 2^15 dalla tabella: */
      sin_w = sinus[zr_w];
      cos_w = sinus[ W_PLUS_90(zr_w) ];

      /* Calcolo della matrice di rotazione e divisione per 2^15 */
      /* (Calcolo all'indietro dei valori di seno/coseno         */
      zwis = (x*cos_w - y*sin_w) >> 15;
      y    = (x*sin_w + y*cos_w) >> 15;
      x    = zwis;
   }
   punkt->x = x;          /* Coordinate indietro   */
   punkt->y = y;
   punkt->z = z;
}


/* Proiezione centrale di un punto */
/*********************************/

VOID pun_zen(punkt, xz, yz, zz)
struct punkt *punkt;       /* Puntatore alla struttura del punto */
WORD xz, yz, zz;           /* Coordinate osservatore             */

{
   REGISTER WORD zwis;

   if (zwis = punkt->z - zz)  /* solo in caso di zwis != 0      */
   {
      punkt->x = (LONG)xz - ( (LONG)zz * (LONG)(punkt->x - xz) )/zwis;
      punkt->y = (LONG)yz - ( (LONG)zz * (LONG)(punkt->y - yz) )/zwis;
      punkt->z = 0;
```

```
    }
    else
    {
        /* Coordinata z del punto uguale a coordinata z osservatore */
        punkt->x = xz;
        punkt->y = yz;
        punkt->z = 0;
    }
}


/* Tracciare (su un piano nascosto) un oggetto sullo schermo */
/**************************************************************/

VOID obj_zeichnen(w, objekt)
struct welt *w;
struct objekt *objekt;     /* Puntatore all'oggetto */
                           /* da tracciare          */

{
    struct punkt *punkte; /* Puntatore alla matrice di punti */
    struct linie *linien; /* Puntatore alla matrice di linee */
    REGISTER UWORD i;
    WORD x1, x2, y1, y2;  /* Coordinate punto finale         */

    if (objekt->sichtbar) /* Disegno solo di un oggetto      */
                          /* visibile                        */
    {
        punkte = objekt->punkte;
        linien = objekt->linien;

        /* Impostazione colori: */
        SetAPen( RastPort[Fen_Nr_v], (LONG)objekt->farbe );

        for (i=0; i < objekt->anz_lin; i++)
        {
            x1 = w->xe_tra + w->xe_ska * punkte[ linien[i].p1 ].x;
            y1 = w->ye_tra - w->ye_ska * punkte[ linien[i].p1 ].y;
            x2 = w->xe_tra + w->xe_ska * punkte[ linien[i].p2 ].x;
            y2 = w->ye_tra - w->ye_ska * punkte[ linien[i].p2 ].y;

            /* Tracciato linee: */
            Move( RastPort[Fen_Nr_v], (LONG)x1, (LONG)y1 );
            Draw( RastPort[Fen_Nr_v], (LONG)x2, (LONG)y2 );
        }
    }
}
```

A questo punto lanciamo il programma: viene aperto un nuovo schermo, cambiati i colori e appare una piccola e graziosa casetta in prospettiva. Proseguiamo: premiamo 10 o 20 volte il tasto "+" (sulla tastiera americana il tasto "]"). La casetta diventa una casa. Con il tasto "-" del tastierino numerico potremmo riportarla alle sue dimensioni originali. Ora che ci troviamo di fronte ad essa, non rinunciamo a girarle attorno: premiamo il tasto di "cursore verso destra".

Ora stiamo ruotando attorno alla casa - oppure è la casa che ruota intorno a noi, o ruota intorno a se stessa? Premiamo più volte lo stesso tasto. Comincia a girarci la testa? Forse sarebbe meglio premere per alcune volte "cursore a sinistra" oppure, ancora meglio: "Enter" (tastierino numerico). Ora la casa è di nuovo ferma.

Premiamo due volte "cursore verso sinistra" e "cursore verso l'alto", nonché "Shift" e "cursore verso destra". Le rotazioni attorno a diversi assi si sovrappongono. Premiamo ora per circa 10 volte il numero "4" della tastiera numerica. Stiamo spostando il centro di rotazione. La casa ruota non più attorno a se stessa, bensì attorno ad un altro punto. Possiamo continuare a provare come vogliamo.

Oltre alle funzioni dei tasti sopraelencati, il programma offre numerose altre possibilità di intervento. (La "n" dopo un tasto significa che si tratta di un tasto situato sul tastierino numerico):

"Tasto destro del mouse"	Posizionamento degli oggetti sul monitor alla posizione indicata dal puntatore del mouse
"Più"	Ingrandimento dell'oggetto
"Meno N"	Riduzione dell'oggetto
"4 N"	Avvicinamento dell'oggetto
"6 N"	Allontanamento dell'oggetto
"8 N"	Allontamento dell'osservatore (punto di fuga)
"2 N"	Avvicinamento dell'osservatore (punto di fuga)
"cursore destra"	Rotazione a destra attorno all'asse y
"cursore sinistra"	Rotazione a sinistra attorno all'asse y
"cursore verso alto"	Rotazione a destra attorno all'asse x
"cursore verso basso"	Rotazione a sinistra attorno all'asse x
"Shift" "cursore destra"	Rotazione a destra attorno all'asse z
"Shift" "cursore sin."	Rotazione a sinistra attorno all'asse z
"Del"	Attivazione/disattivazione sistema di coordinate
"0 N"	Attivazione/disattivazione trasformazione sistema di coordinate
"Help"	Fine delle rotazioni, trasformazioni riportate ai valori originali
"Enter N"	Fine di tutte le rotazioni
"Punto N"	Attivazione/disattivazione del Double Buffering
"Esc"	Termine del programma

Sarà tuttavia necessario fare attenzione a non usare il tasto sinistro del mouse, dal momento che potrebbe succedere che il programma non reagisca più agli input che gli vengono dati (vedremo subito perché). Nel caso in cui ciò fosse già successo, teniamo premuto ripetutamente il pulsante sinistro del mouse finché il programma non sarà di nuovo attivo.

Naturalmente questo programma ci divertirà molto al suo apparire. Tuttavia, più avanti, la casa non ci piacerà più: vorremo anche un'automobile, una bicicletta, oppure un'intera città. Da questo punto di vista, il programma è estremamente flessibile. Esso può

maneggiare, senza modifche, un numero quasi a piacere di oggetti. Ogni oggetto potrà avere il proprio colore, potrà essere visibile o invisibile ecc. Ogni oggetto potrà essere composto da un numero di punti e di linee a piacere. L'unica cosa che dovremo fare per rappresentarlo è quella di inserire i dati necessari (descrizione dell'oggetto, punti, linee) e ricompilare il programma. Diversamente, sarà possibile memorizzare i dati (per esempio su disco) e mantenere invariato il programma vero e proprio. Attualmente il piccolo mondo del nostro computer è composto da due oggetti: un sistema di coordinate e una casa. Inoltre, non sarà possibile trasformare nè tutti gli oggetti insieme, nè i singoli oggetti. Ciò è dovuto al fatto che i dati di trasformazione sono presenti una volta sola nella struttura di tale mondo. Se inseriamo tali dati (rotazione, traslazione, ecc.) in aggiunta alla struttura di ogni oggetto, cosa che, dal punto di vista del programma, non è assolutamente un problema, potremo ruotare, ingrandire ecc. ogni oggetto singolarmente preso e separato dagli altri. Le coordinate di ogni punto dell'oggetto si riferiranno quindi solo al sistema di coordinate proprio dell'oggetto stesso. Ogni sistema di coordinate dell'oggetto avrà la propria posizione, sempre variabile, nel suo mondo. Naturalmente tale mondo possiede a sua volta le trasformazioni, e potrà venire ruotato, ingrandito, ecc. nell'insieme.

Passiamo ora alla descrizione del programma:

Ciò che andrebbe fatto prima di tutto in un programma per Amiga, cioè l'apertura e inizializzazione dei diversi componenti del sistema, è già stato appreso in un capitolo precedente. Si trattava infatti delle library. Il programma ha bisogno, oltre alla Exec-Library già aperta, anche della Intuition-Library e della Graphics-Library. Per precauzione, in caso di futuri ampliamenti di tale programma, apriamo anche la MathFFP-Library, anche se per il momento non utilizziamo le librerie matematiche veloci.

Quanto segue è già noto: l'apertura di un nuovo schermo con una finestra. Tuttavia, in questo caso, ne vengono aperti due contemporaneamente. Ciò ha a che vedere con il cosiddetto Double Buffering, di cui ci occuperemo fra poco. Non meravigliamoci se non vediamo nessuna finestra. Abbiamo scelto l'opzione BORDERLESS. Di conseguenza le finestre non sono dotate di contorni. Inoltre non troveremo nessun gadget nelle finestre. Anch'essi sono completamente invisibili. Tramite SetRGB4( ) vengono impostati i colori. E'solo a questo punto che incontriamo la chiamata di programma vera e propria: **do_program( ).**

Prima di addentrarci nel programma vero e proprio, approfondiamo brevemente quanto si trova sotto il titolo "Variabili e strutture globali". Si tratta di una tabella gigantesca: la tabella dei seni. Per le matrici di rotazione abbiamo bisogno di calcolare seni e coseni di angoli a piacere. Ciò, naturalmente, potrebbe venire effettuato in maniera molto semplice tramite le funzioni della Library matematica dell'Amiga. Ma quanto impiegheremmo? Noi vogliamo ruotare il nostro oggetto in maniera relativamente veloce sullo schermo. Quindi, poiché nel programma abbiamo a che fare solo con angoli interi (cioè senza virgola) creiamo semplicemente una tabella con 360 valori (per gli angoli da 0 a 359). Ogni valore indica il seno di un angolo ampio un ben determinato numero di gradi. Nel caso in cui il programma avesse bisogno del seno, per esempio, di 157°, esso andrà a consultare tale tabella e, nel giro di microsecondi, si otterrà il seno desiderato. Non è necessario nessun altro calcolo.

Con la stessa tabella sarà possibile determinare anche il coseno di un angolo. Forse ricordiamo ancora dalla scuola la seguente equazione:

cos(a) = sin(a+90)

oppure, in radianti:

cos(a) = sin(a+PI/2)

Il coseno di 100 gradi è quindi uguale al seno di 100+90 = 190°. Aggiungere 90 all'angolo, controllare nella tabella del seno e il gioco è fatto. L'unica cosa a cui sarà necessario fare attenzione è che l'angolo non potrà mai diventare maggiore di 359°, dal momento che ciò supererebbe la capacità della tabella. Non è tuttavia difficile evitare ciò, dal momento che 360° sono una rotazione completa, cioè corrispondono a 0°. Quindi, se un angolo diventa maggiore di 359°, togliamo semplicemente 360.

La tabella avrebbe anche potuto essere molto più piccola. Senza grossi calcoli, infatti, avremmo potuto inserire i valori del seno da 0 a 179, addirittura da 0 a 89 gradi. Ma la memoria dell'Amiga è grande, e il tempo di calcolo è sempre prezioso.

Come memorizzare allora i valori del seno? In conclusione tutti i valori sono contenuti fra -1 e 1. I calcoli con la virgola portano via troppo tempo. Soluzione: nella tabella i valori di seno non sono memorizzati direttamente, ma moltiplicati per $2^{15}=32768$. Ciò corrisponde, nel sistema binario, a uno spostamento di 15 Bit verso sinistra. Con ciò tali valori rientrano perfettamente in una parola (il Bit più alto, cioè il quindicesimo, serve come bit per il segno). In seguito, nel programma, al momento delle moltiplicazioni con le coordinate di un punto, si procederà all'inverso, (anche se senza arrotondamenti) cioè (dopo la moltiplicazione) effettueremo una divisione per $2^{15}=32768$ (spostamento di 15 bit verso destra). Questo spostamento può venire eseguito molto velocemente e senza sforzi dal processore (anche in C), con un solo comando. Non è quindi necessaria nessuna lunga divisione. Seguono quindi le definizioni di struttura per un punto, una linea, un oggetto e l'intero mondo (infatti ogni punto, linea ecc. è una struttura nella quale sono presenti le coordinate, cioè i numeri dei punti finali). All'interno della struttura dell'oggetto si trovano due puntatori. Tutti i punti e tutte le linee di un oggetto sono contenuti in due grandi matrici (ogni matrice è composta quindi da numerosissime strutture di punto e di linea). I puntatori alla struttura dell'oggetto puntano proprio a queste matrici. Inoltre in questa struttura è contenuta l'indicazione del numero di punti e linee di cui l'oggetto si compone, ed il suo colore (è possibile inserire anche un nome). Inoltre è annotato anche se l'oggetto è visibile sullo schermo e se deve essere soggetto alle trasformazioni nello spazio. In ogni struttura spaziale si trova inoltre un puntatore ad un'altra matrice. Le strutture di tutti gli oggetti sono infatti contenute anch'esse in una matrice. Nella struttura spaziale riconosceremo finalmente i valori per le diverse trasformazioni, le coordinate dell'osservatore e le trasformazioni sul piano per l'adattamento allo schermo.

Esistono ancora altre due matrici che meritano di essere citate. Si tratta di una seconda matrice dell'oggetto e di una seconda matrice dei punti. (I loro nomi sono:

trans_obj[ ] (o anche _trans_obj[ ]) e _trans_pun[ ]). Esse sono strutturate esattamente come quelle precedentemente descritte e hanno le stesse dimensioni. All'inizio del programma esse vengono riscritte con gli stessi valori. Tuttavia, quando un oggetto deve venire trasformato e proiettato con tutti i propri punti, è solo in questo secondo gruppo di matrici che vengono memorizzati i risultati intermedi di queste trasformazioni. In pratica, per il calcolo serve solo il secondo gruppo di matrici. In questo modo vengono conservate le coordinate originali del punto.

Le relazioni tra le varie strutture vengono rappresentate nella figura seguente:

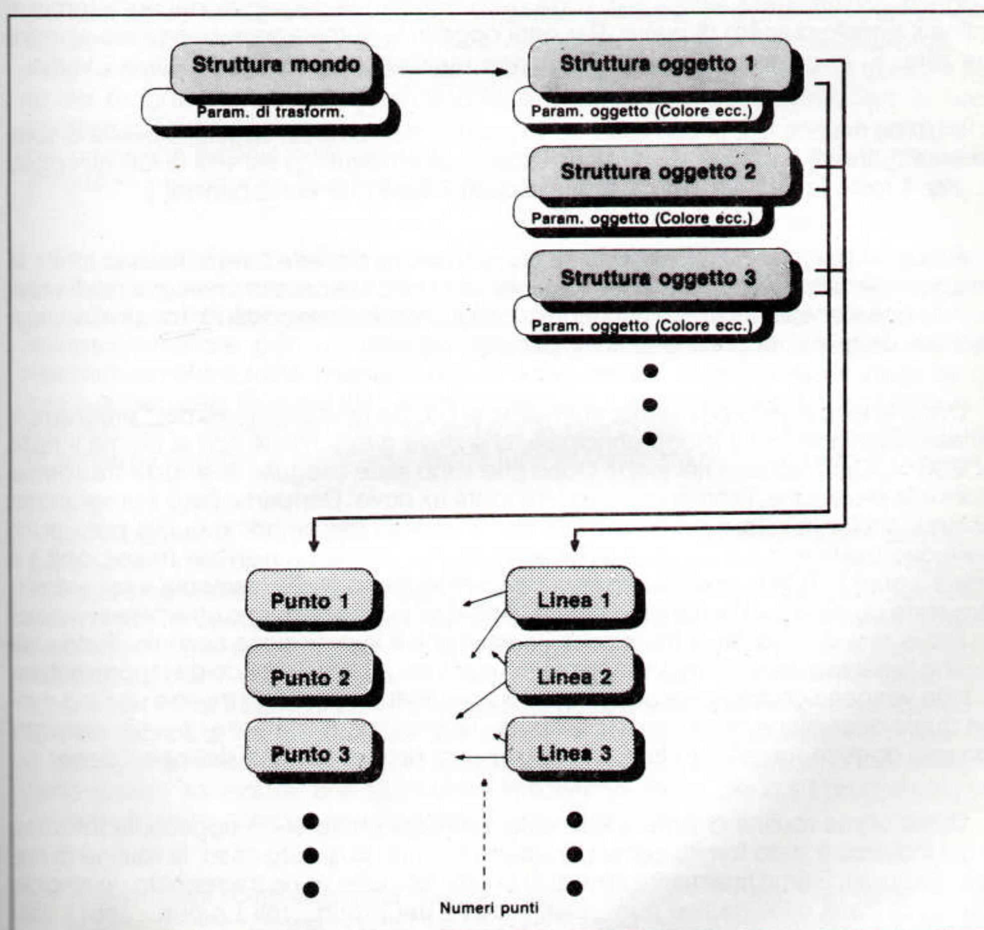


Fig. 4.24 Organizzazione dei dati del programma in C

Resta solo il fatto che tutte queste strutture non sono inizializzate (cioè riempite di dati). L'inizializzazione ha luogo nella funzione **schaffe_welt( )** (crea-mondo), chiamata per seconda nel **do_program( )**.

Prima di cercare di comprendere il procedimento di questa funzione, dovremo occuparci dell'input dei dati tridimensionali (vedasi anche sotto il titolo "Approntamento di tutte le strutture dati"). In esso troviamo dapprima una matrice di strutture che descrive i nuovi oggetti (**new_object[ ]**). Nel caso in cui volessimo modificare il sistema di coordinate della casa, è qui che immetteremmo per ogni oggetto il numero di punti e di linee di cui esso è costituito, che colore ha e come si chiama. Il numero degli elementi qui inseriti determina da quanti oggetti è costituito il nostro mondo (in questo caso 2), e la sequenza determina il numero di oggetti.

Quindi troviamo la matrice dei punti **_punkte[ ]**. Essa contiene tutti i punti di ciascun oggetto, ordinati nella stessa sequenza degli oggetti. La sequenza dei punti fornisce infine il singolo numero di punto. Per ogni oggetto la numerazione ricomincia sempre da zero. E' in questa matrice che vengono registrate le coordinate di tutti i vertici.

L'ultima matrice che ci interessa, se vogliamo inserire nuovi oggetti, è quella di tutte le linee **_linien[ ]**. In essa sono memorizzati i numeri di tutti gli estremi di tutti gli oggetti. Per il resto la sua struttura è simile a quella della matrice _punkte[ ].

A questo punto si comprenderà meglio la funzione **schaffe_welt.** In essa infatti la matrice normale dell'oggetto viene reimpita con i valori necessari (mentre a quella della funzione viene attribuito un puntatore). I valori inseriti sono costituiti, fra gli altri, dagli indirizzi degli inizi di punti e di linee per ogni oggetto.

Ora che tutto è stato preparato, ritorniamo al programma principale **do_program( )**. In esso comincia infatti il loop principale, che deve girare finché non si preme il tasto <ESC>. Cosa accade nel loop? Dopo che sono state eseguite le singole trasformazioni e la proiezione, l'immagine viene tracciata ex novo. Dapprima però il programma chiama anche **init_ausgabe( )**. E' qui che le matrici per oggetti e quelle per i punti vengono trasferite nelle matrici precedentemente descritte, chiamate **trans_obj[ ]** e **trans_pun[ ]**. Tutti i calcoli successivi fanno riferimento esclusivamente a tali matrici. A questo punto si può veramente partire. Una dopo l'altra vengono chiamate le funzioni per la messa in scala, la traslazione, la rotazione e la proiezione centrale. Tutte eseguono le necessarie moltiplicazioni di matrici con tutti i punti del mondo da rappresentare. E tutte vengono chiamate da una stessa routine: **welt_verteiler( )** mentre un Flag indica quale operazione deve venire effettuata. Tale routine a sua volta chiama, per ogni singolo oggetto, un'altra routine, che si occupa dell'esecuzione dell'operazione.

Quest'ultima routine chiama a sua volta, per ogni punto di un oggetto, la funzione il cui indirizzo è stato fornito come parametro (quindi, in questo caso, la routine di trasformazione). Siamo finalmente arrivati al punto nel quale viene trasformato un singolo punto. Si tratta delle routine **pun_ska( ), pun_tra( ), pun_rot( )** e **pun_zen( )**. Qui hanno luogo veramente le moltiplicazioni delle matrici e le formule non ci giungeranno nuove.

Quando infine tutti i punti di tutti gli oggetti sono stati trasformati, il programma ritorna al **do_program( )**. Quando sono state eseguite tutte le trasformazioni sarà possibile disegnare. Quindi cancelliamo tutto ciò che è sullo schermo e cominciamo.

Anche l'operazione di disegno vera e propria viene effettuata tramite il **welt_verteiler( )**. Dopo ciò, viene selezionata la funzione **obj_zeichnen( )**, che fornisce ad un oggetto il colore prescelto.

Siamo appena ritornati al **do_program( )** e incontriamo una riga di commento che dice: "portare davanti la finestra coperta", seguita da una chiamata ScreenToFront( ), la quale fa apparire un nuovo schermo davanti al precedente. Cosa accade ora? Senza averlo nemmeno notato, abbiamo applicato in diversi punti del programma un tecnica il cui effetto è chiaramente visibile. Rilanciamo il programma e premiamo uno o più tasti del cursore affinché l'immagine cominci a ruotare. Ora premiamo brevemente il tasto di punto della tastiera numerica e vedremo immediatamente la differenza: l'immagine o gli oggetti cominciano a sfarfallare. Ciò è dovuto al fatto che in qualche punto del programma è necessario cancellare lo schermo al fine di tracciare la nuova immagine. Prima però che la nuova immagine sia completa, trascorre un certo tempo nel quale non avremo nulla sullo schermo, oppure solo una parte dell'oggetto stesso e l'effetto indesiderato di questi passaggi è lo sfarfallamento.

Per ovviare a questo effetto esiste il trucco del Double Buffering, cioè, in questo caso, del disegno nascosto. All'inizio abbiamo già menzionato il fatto che all'apertura di ciascuna finestra abbiamo aperto due Screen. Tuttavia è possibile osservare solo lo schermo anteriore, per cui, quando disegnamo, procediamo come segue: mentre la vecchia immagine resta invariata sullo schermo visibile, quello invisibile viene cancellato e ridisegnato. Quando tutto è pronto, facciamo apparire tale schermo finora invisibile con uno "ScreenToFront( )" e avremo evitato lo sfarfallamento. A questo punto, lo schermo che ora è invisibile subisce tutte le trasformazioni del caso e così via.

Nel programma, le variabili **Fin_Nr_s** e **Fin_Nr_v** indicano i numeri dello schermo visibile o nascosto (s per visibile e v per nascosto). Questo numero determina quale Rasterport viene trasferita ad ogni operazione di disegno. Al momento del cambio dell'immagine cambiamo anche i numeri in queste due memorie. E' tuttavia possibile interrompere questa sequenza con la pressione del punto decimale, dopodiché potremo mettere dietro lo schermo che è davanti e vedere il secondo schermo. Certo, forse avremmo potuto evitare di aprire due schermi per il Double buffering, modificando solo il Bitmap della finestra, ma sarebbe stato più complesso. Continuiamo quindi così e facciamo attenzione al seguente svantaggio derivante: non dovremo mai toccare il tasto sinistro del mouse, che attiverebbe lo schermo errato, per cui il programma verrebbe a trovarsi completamente staccato dai nostri input.

Passiamo alla scansione della tastiera. Se al momento l'oggetto non viene ruotato, sarà possibile attendere con **Wait( )** la pressione di un tasto (nella struttura newwindow avevamo indicato di segnalarci qualunque pressione di un tasto, da tastiera o da mouse). Diversamente, saltiamo l'attesa e determiniamo immediatamente con **Get_Msg( )** se è arrivata qualche "notizia" (tasto). La "notizia" è composta dalla struttura IntuiMessage, cioè:

Class	Tipo di messaggio (IDCMP Flag)
Code	Codice RAW del tasto

Qualifier Tasto premuto contemporaneamente (Shift ecc.)
Mouse_x Posizione x del Mouse (solo per tasti del mouse)
Mouse_y Posizione y del Mouse (solo per tasti del mouse)

Con **ReplyMsg( )** rinviamo il messaggio. Quindi eseguiamo i compiti richiesti con semplici modifiche delle variabili. Se non si deve attendere la pressione di un altro tasto, il Flag è sempre uguale a zero. Si tratta del caso in cui è stato premuto un tasto valido e l'oggetto sta ruotando. Con <ESC> la routine ha termine e rinvia un codice di errore a **main( )**.

4.5.2 Rotazione attorno ad un asse a piacere nello spazio

A questo punto è possibile far ruotare i nostri oggetti attorno a tutti i tre gli assi delle coordinate. Al fine però di ottenere un effetto globale, siamo interessati alla rotazione attorno ad un asse a piacere nello spazio. Può sembrare complicato ma non lo è, dal momento che dobbiamo solo rivolgerci alle già note matrici di trasformazione tridimensionale. Verificheremo ancora una volta l'utilità che ha avuto per noi l'adozione della scrittura in forma di matrici.

Dapprima però dobbiamo definire l'asse attorno al quale la nostra immagine deve ruotare. Facciamolo con l'equazione vettoriale di una retta, a noi già nota:

$$P = P_0 + s*R$$

cioè, in parametri:

$$x = x_0 + s*R_x$$
$$y = y_0 + s*R_y$$
$$z = z_0 + s*R_z$$

dove:

P — un punto calcolato della retta, con le coordinate x,y,z
P_0 — un punto a piacere della retta, con le coordinate x_0,y_0,z_0
R — vettore di direzione con componenti R_z, R_y, R_z
s — fattore di allungamento del vettore di direzione

La nostra strategia corrisponde esattamente alla rotazione attorno ad un punto a piacere sul piano (ved. in precedenza). Cercheremo quindi di portare a coincidere ad un asse di coordinate l'asse attorno al quale deve venire effettuata la rotazione, e ci riusciremo spostando la retta in modo che attraversi l'origine delle coordinate con un punto a piacere. Successivamente ruotiamola attorno all'asse x, in modo che venga a giacere completamente sul piano x-z. Ha quindi luogo una rotazione attorno all'asse y, in modo che la retta coincida con l'asse z. A questo punto siamo in grado di eseguire veramente la rotazione desiderata. Ruotiamo quindi l'oggetto con l'angolo desiderato attorno all'asse z. Infine dovremo effettuare all'indietro ambedue le rotazioni attorno all'asse x ed y. La figura 4.25 chiarirà meglio il procedimento.

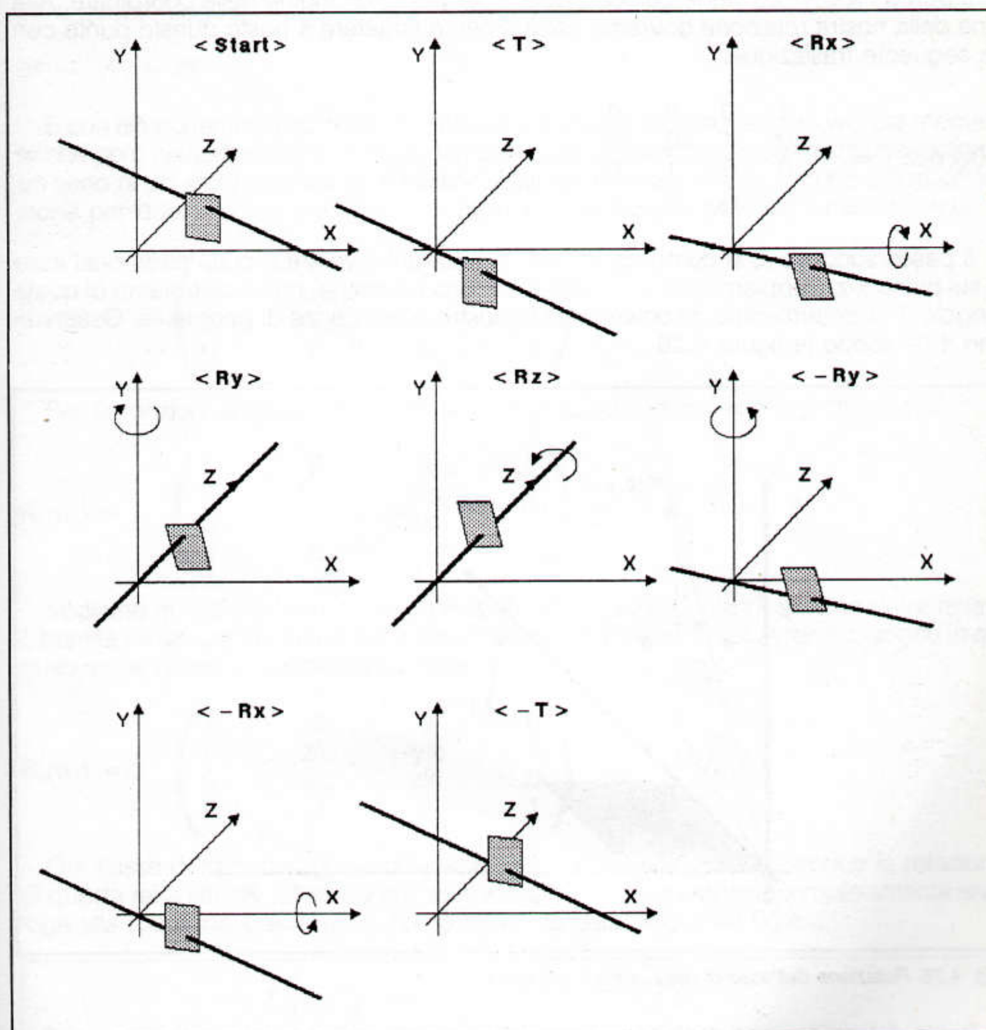


Fig. 4.25 Passaggi per la rotazione attorno ad un asse a piacere

Tutto inizia con la translazione della linea all'origine del sistema di coordinate, che otterremo con la matrice di traslazione:

$$T(-x_0,-y_0,-z_0) = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{matrix}$$

Il punto $P_0(x_0,y_0,z_0)$ viene quindi spostato nel punto di origine delle coordinate. Alla fine della nostra rotazione dovremo naturalmente rimettere a posto questo punto con la seguente traslazione:

$$T(x_0,y_0,z_0) = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{matrix}$$

Il passo successivo si complica un po': la retta deve venire ruotata attorno all'asse x sul piano x-z. Sappiamo come far effettuare una rotazione, ma ci chiediamo di quale angolo. Per determinarlo, rispolveriamo le nostre conoscenze di geometria. Osserviamo a tal scopo la figura 4.26.

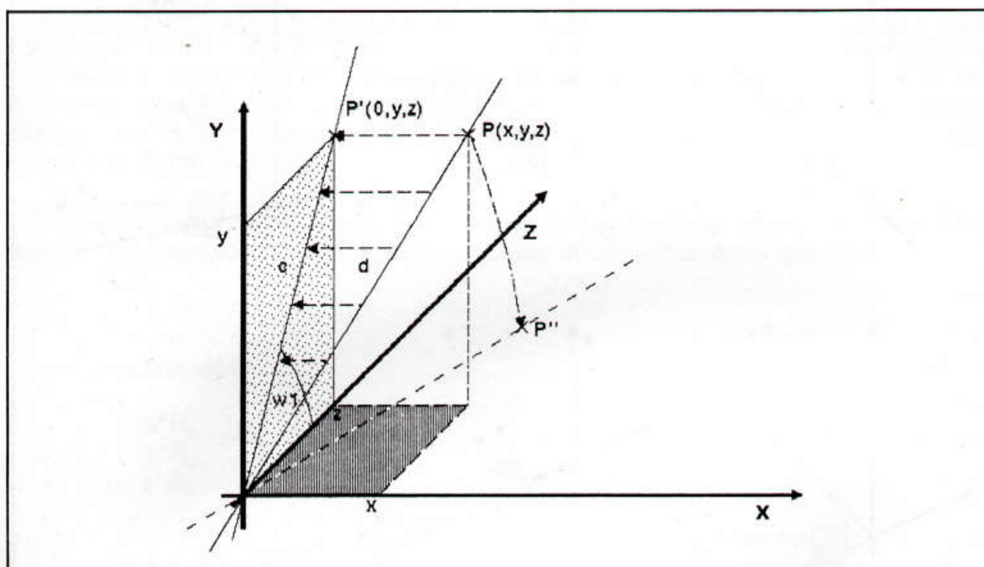


Fig. 4.26 Rotazione dell'asse di rotazione sul piano x-z

Per la determinazione dell'angolo w_1, proiettiamo dapprima la retta sul piano y-z. Osserviamo però che qui non si ha ancora nessuna rotazione. Abbiamo invece a che fare con una semplice proiezione parallela. Prendiamo un punto a piacere P(x,y,z) della retta. Tracciamo quindi da questo punto una parallela all'asse x. Il punto nel quale la parallela interseca il piano y-z è il punto proiettato P' con le coordinate (0,y,z). L'angolo w_1, necessario al fine di ruotare la retta sul piano x-z, è quello compreso fra la retta proiettata e l'asse z (ved. figura). Ora è senz'altro più semplice calcolare tale angolo.

Secondo la definizione di angolo di un triangolo rettangolo, vale quanto segue (c è la distanza tra l'origine delle coordinate ed il punto proiettato P'):

$\sin(w_1) = y/c$
$\cos(w_1) = z/c$

Ma c non ci è sconosciuto, in quanto, secondo il teorema di Pitagora, possiamo scrivere quanto segue:

$$c^2 = y^2 + z^2$$

E con ciò potremmo calcolare, in maniera puramente teorica, l'angolo w_1. Dal momento che però necessiteremmo di molto tempo di calcolo (dovremmo calcolare una radice e un seno di un arco) cerchiamo di evitarlo. Sviluppiamo quindi una matrice di trasformazione per questa rotazione attorno all'asse x. Com'è noto, essa sarà normalmente:

$$R_x(w_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(w_1) & \sin(w_1) & 0 \\ 0 & -\sin(w_1) & \cos(w_1) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Per le funzioni angolari utilizziamo le due equazioni di cui sopra e otteniamo:

$$R_x(w_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & z/c & y/c & 0 \\ 0 & -y/c & z/c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Vediamo quindi che non abbiamo bisogno di un angolo, ma dobbiamo determinare c tramite estrazione di radice. La trasformazione all'indietro di cui avremo bisogno in seguito viene quindi determinata facilmente:

$$R_x(w_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & z/c & -y/c & 0 \\ 0 & y/c & z/c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ora l'asse di rotazione giace sul piano x-z. Il passo successivo sarebbe la rotazione di questa retta attorno all'asse y sull'asse z. Ciò accadrà in maniera completamente analoga alla rotazione precedente, come rilevabile dalla seguente figura:

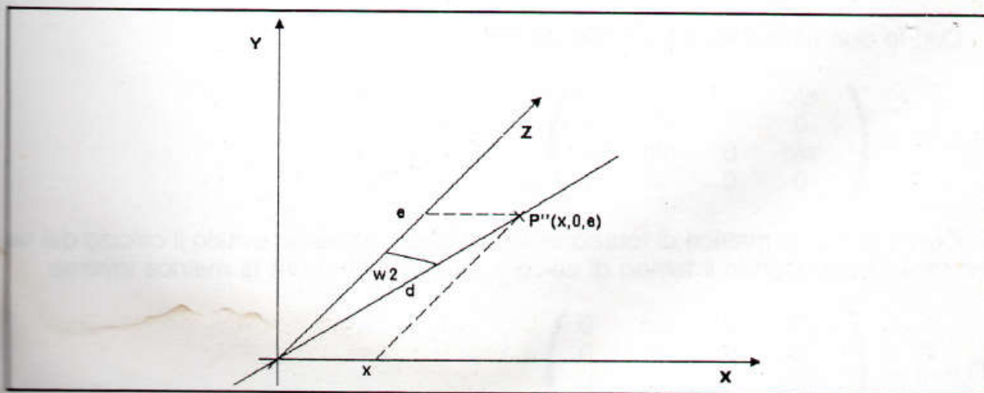


Fig. 4.27 Rotazione dell'asse di rotazione sull'asse z

Anche in questo caso ci serve l'angolo di rotazione w_2.

Al momento della rotazione, eseguita in precedenza, attorno all'asse x, è già stata determinata la coordinata x del punto P. Inoltre è rimasta invariata la lunghezza del segmento d tra l'origine e P. Tale lunghezza può venire determinata (teorema di Pitagora nello spazio, osserviamo ancora la fig. 4.26) con:

$$\begin{aligned} d^2 &= x^2 + y^2 + z^2 \\ &= x^2 + c^2 \end{aligned}$$

(Anche in questo caso calcoliamo d estraendo la radice dalla parte destra dell'equazione). Potremo determinare d anche dalla variabile c, a noi nota dopo l'ultima rotazione. La coordinata y del punto ruotato P'' è naturalmente uguale a 0 (infatti esso giace sul piano x-z). Determiniamo la coordinata z applicando ancora una volta il teorema di Pitagora (ved. figura):

$$\begin{aligned} e^2 &= d^2 - x^2 \\ &= x^2 + c^2 - x^2 \\ &= c^2 \end{aligned}$$

Quindi e è uguale a c, precedentemente determinato, cosa che ci risparmia ulteriori calcoli. Al fine di trovare ora la matrice di rotazione desiderata, applichiamo quanto segue, come da fig. 4.27:

$$\begin{aligned} \sin(w_2) &= x/d \\ \cos(w_2) &= c/d \end{aligned}$$

La matrice di rotazione normale per una rotazione attorno all'asse y dovrebbe già esserci nota:

$$R_x(w_2) = \begin{pmatrix} \cos(w_2) & 0 & -\sin(w_2) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(w_2) & 0 & \cos(w_2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Con le due ultime equazioni otteniamo:

$$R_y(w_2) = \begin{pmatrix} c/d & 0 & -x/d & 0 \\ 0 & 1 & 0 & 0 \\ x/d & 0 & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Così è pronta la matrice di rotazione. Come sopra abbiamo evitato il calcolo del seno di w_2 risparmiando il tempo di calcolo. Determiniamo ora la matrice inversa:

$$R_x(w_1) = \begin{pmatrix} c/d & 0 & x/d & 0 \\ 0 & 1 & 0 & 0 \\ -x/d & 0 & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Finalmente l'asse di rotazione giace sull'asse z del sistema di cocordinate. Ora l'oggetto può venire ruotato normalmente con la matrice di rotazione $R_z(a)$ attorno all'asse z. Dopo ciò, invertiamo tutte le rotazioni, a tal scopo possiamo riassumere tutta l'operazione in una lunga formula:

$R_{asse}(a)$ =

$T(-x_0,-y_0,-z_0)$ * $R_x(w_1)$ * $R_y(w_2)$ *
$R_z(a)$ *
$R_y(-w_2)$ * $R_x(-w_1)$ * $T(x_0,y_0,z_0)$

Ricordiamo che gli angoli w_1 e w_2 non dovranno venire calcolati, le coordinate x_0, y_0 e z_0 indicano un punto dell'asse di rotazione e l'angolo a è l'angolo di rotazione effettivo attorno all'asse.

E naturalmente saremo ora in grado di determinare da tutte queste moltiplicazioni di matrici una singola matrice $R_{asse}(a)$, che esegua tutte insieme le varie rotazioni e inversioni di rotazioni. Per tale matrice (o almeno per il prodotto di cui sopra) dovremo moltiplicare ciascun punto da ruotare.

CAPITOLO 5

Linee e superfici nascoste
Il problema e le soluzioni

A questo punto siamo in grado di definire immagini spaziali complesse, che devono venire spostate nello spazio, ingrandite e fatte ruotare attorno ad assi a piacere, devono venire proiettate sullo schermo tenendo conto del punto di fuga, dell'osservatore e via di seguito. Una cosa ci ha sicuramente disturbato finora: tutti gli oggetti, sia piccoli, che grandi, erano trasparenti. Gli angoli posteriori, che sarebbero coperti, erano anch'essi visibili, come se ci trovassimo di fronte a modelli in filo metallico (wire frame).

La rappresentazione di tali immagini tracciate coerentemente come modelli di filo è la maniera più semplice di proiezione spaziale, dal momento che non ci interessa quali linee e punti dobbiamo tracciare e quali no. Tuttavia, per ottenere una impressione ancora migliore di una immagine spaziale, dovremmo tentare di prendere noi l'iniziativa.

Esistono gli algoritmi più svariati per le applicazioni più diverse, atti alla copertura delle linee e superfici nascoste. Di seguito impareremo anche questi.

5.1 Ancora più semplice: Linee nascoste in funzioni tridimensionali

5.1.1 Il principio

Nel presente e nei Capitoli seguenti vogliamo cercare di ottenere due vantaggi in una volta: uno di questi sarà di soddisfare il problema delle linee coperte, l'altro ci offrirà una interessante possibilità di produrre immagini meravigliose con l'aiuto delle funzioni tridimensionali.

Le funzioni bidimensionali ci sono certamente già note. Chi non ha mai visto almeno una volta una equazione con due incognite (una equazione della retta, per es.)? In generale una funzione simile può venire espressa come segue:

$$y = f(x)$$

A destra abbiamo una espressione a piacere con tutti i calcoli possibili e impossibili, le funzioni di seno e logaritmo ecc. che contengono tutte una sola incognita: x. Il risultato di un simile termine diventa poi il valore che verrà inserito per y. Il risultato dipenderà dal valore che inseriremo per x. Si usa dire: y dipende da x, o è funzione di x. Un esempio semplice è:

$$y = x$$

In questo caso y assume lo stesso valore di x.

Una tale funzione potrà venire rappresentata anche graficamente. In questo caso si utilizzerà un sistema bidimensionale di coordinate o piano cartesiano. Su di un asse conteremo il valore y, sull'altro il valore x. Il risultato sarà quindi la curva sopracitata

(nel caso suddetto una diagonale che passa attraverso l'origine). Ma per quale motivo vi racconto ciò? Lo sappiamo già benissimo. Ma ora diventa più interessante: ci sono funzioni che non vanno d'accordo con i due parametri x e y. In tali funzioni entra in gioco un terzo parametro. Di conseguenza tali funzioni hanno la seguente forma:

$y = f(x,z)$

Il valore della funzione y dipende anche dalle due variabili x e z (in alcuni testi viene indicato anche $z = f(x,y)$). Per la rappresentazione grafica di tali funzioni, non riusciremmo a procedere con la semplice tecnica di disegno bidimensionale. Inseriremo quindi un asse z che ci apre alla dimensione dello spazio. Dovremo quindi ritornare alle formule tridimensionali che abbiamo appreso nei Capitoli precedenti.

Infatti tale applicazione non si limita solo al piacere dell'osservazione, ma rende possibile rappresentare in un solo diagramma delle correlazioni complesse, che dovrebbero venire rilevate, in altre maniere, da numerose immagini singole. Non solo le equazioni matematiche si adattano alla maniera qui esposta, anche le tabelle statistiche sono funzioni matematiche rigorose che possono venire rappresentate in questo modo. L'unica differenza è che con le tabelle non vengono calcolati i valori singoli, come invece accade con le equazioni.

Prendiamo un esempio: supponiamo che ci interessino il volume di affari annuo della nostra impresa (o del nostro patrimonio familiare). Nulla è più facile di ciò: appronteremo dapprima un tabella, nella quale inseriremo l'anno e la cifra relativa. A questo punto ha luogo il disegno con gli anni sull'asse x orizzontale e con le cifre sull'asse y verticale. Ogni valore y (cifra) è perciò dipendente dal valore x corrispondente (anno). I diagrammi tridimensionali potranno essere necessari se si desiderano tracciare anche le cifre delle imprese concorrenti o dei membri della famiglia. Tracceremo nello spazio un ulteriore asse z, sul quale ordineremo le singole voci da calcolare. In questa maniera otterremo una visione ottimale della situazione generale e i figli o il coniuge potranno vedersi ridurre tempestivamente il denaro per le piccole spese.

A questo punto dovremo rivolgere di nuovo la nostra attenzione al problema principale. Nel nostro caso apprenderemo le tecniche, sulla base delle equazioni matematiche (funzioni), con le quali si sono ottenuti i valori delle funzioni. Ciò ci farà risparmiare l'esecuzione delle tabelle.

Tramite l'esecuzione di una cosiddetta tabella di valori, otterremo il grafico (o la curva) di una funzione e su tale tabella ordineremo tre colonne per i valori di x, y e z. Nel nostro esempio si sono manipolate in primo luogo le funzioni seguenti:

$y = f(x,z) = x^2 + z^2$

Come possiamo a realizzare nel programma il calcolo di questa funzione nel modo più efficace? Se dovessimo rappresentare solo una funzione con due incognite (cioè $y = f(x) = x^2$), il problema sarebbe veramente semplice da risolvere. Il nostro compito sarebbe quello di incrementare il valore x proveniente da un determinato punto, se-

guendo passi ben determinati, in un loop FOR...NEXT fino ad un valore finale. Ad ogni ciclo del loop si calcolerebbe il valore y mancante dipendente dal valore x. Entrambi i valori ottenuti potranno quindi venire visualizzati molto semplicemente sullo schermo. Ecco lo schema di un plotter di funzione bidimensionale:

```
FOR x = <valore iniziale di x> TO <valore finale> STEP <ampiezza passo>
  y = f(x)
  PLOT x,y
NEXT x
```

Con le funzioni tridimensionali, invece, abbiamo due variabili x e z, dalle quali dovrà venire determinata y, la terza. In questo caso ci aiuteremo con due loop FOR...NEXT nidificati. L'uno incrementa il valore x, l'altro il valore z:

```
FOR z = <val. iniz. di z> TO <val. fin. di z> STEP <amp. passo z>
  FOR x = <val. iniz. di x> TO <val. fin. di x> STEP <amp. passo x>
    y = f(x,z)
    PLOT x,y,z
  NEXT x
NEXT Z
```

In questo caso avremo applicato z nel loop esterno e x in quello interno. Questo procedimento non è indispensabile, ma spesso viene raccomandato per tracciare oggetti tridimensionali.

Il comando PLOT x,y,z sarà difficile da trovare in un manuale del linguaggio Basic. Si dovrà prendere in considerazione un punto tridimensionale, tenendo conto di tutte le altre trasformazioni, da proiettare sul piano tramite proiezione centrale. Il seguente programma in Basic chiarirà le cose:

```
' ********************************
' **                            **
' **  Funzioni tridimensionali 1 **
' **                            **
' ********************************

PI = 3.141593

SCREEN 2,640,200,2,2             ' Nuovo schermo
WINDOW 4,,(0,0)-(631,186),0,2 ' Nuova finestra

PALETTE 0,  0,  0,  0            ' Impostazione colori
PALETTE 1, .8,  0,.93
PALETTE 2,.47,.87,  1
PALETTE 3,  1, .6,.67

COLOR 2
```

```
' Parametri di transformazione:
' Scala:

sx = 40
sy = 6
sz = 20

' Translazione:

tx = 0
ty = 0
tz = 0

' Rotazione:

rx = 15
ry = -45
rz = 0

' Osservatore:

bx = 0
by = 0
bz = -300

' Translazione piano:

tex = 300
tey = 90

' Costanti per le rotazioni:

rx = rx*PI/180
ry = ry*PI/180
rz = rz*PI/180

si.x = SIN(rx) : co.x = COS(rx)
si.y = SIN(ry) : co.y = COS(ry)
si.z = SIN(rz) : co.z = COS(rz)

A = co.y * co.z
B = co.y * si.z
C = -si.y
D = si.x*si.y*co.z - co.x*si.z
E = si.x*si.y*si.z + co.x*co.z
F = si.x*co.y
G = co.x*si.y*co.z + si.x*si.z
H = co.x*si.y*si.z - si.x*co.z
I = co.x*co.y
```

```
' Valori di inizio/fine/ampiezza passi:

sta.x = 3 : en.x = -3 : ste.x = -.01
sta.z = 4 : en.z = -3 : ste.z = -.5

' Funzione:
DEF FNfun(x,z) = x*x - z*z

' Disegno:

FOR z=sta.z TO en.z STEP ste.z
  FOR x=sta.x TO en.x STEP ste.x

    ' Calcolo valore della funzione:

    y = FNfun(x,z)

    ' Trasformazioni:

    xt1 = sx*x + tx              ' Scalatura
    yt1 = sy*y + ty              ' e translazione
    zt1 = sz*z + tz

    xt2 = xt1*A + yt1*B + zt1*C ' Rotazione
    yt2 = xt1*D + yt1*E + zt1*F
    zt2 = xt1*G + yt1*H + zt1*I

    ' Proiezione centrale:

    zwis = zt2 - bz
    xe = bx - bz * (xt2-bx)/zwis
    ye = by - bz * (yt2-by)/zwis

    ' Tracciamento del punto:

    PSET (tex + xe, tey - ye)

  NEXT x
NEXT z

WHILE (INKEY$ = "")
WEND

WINDOW CLOSE 4
SCREEN CLOSE 2
```

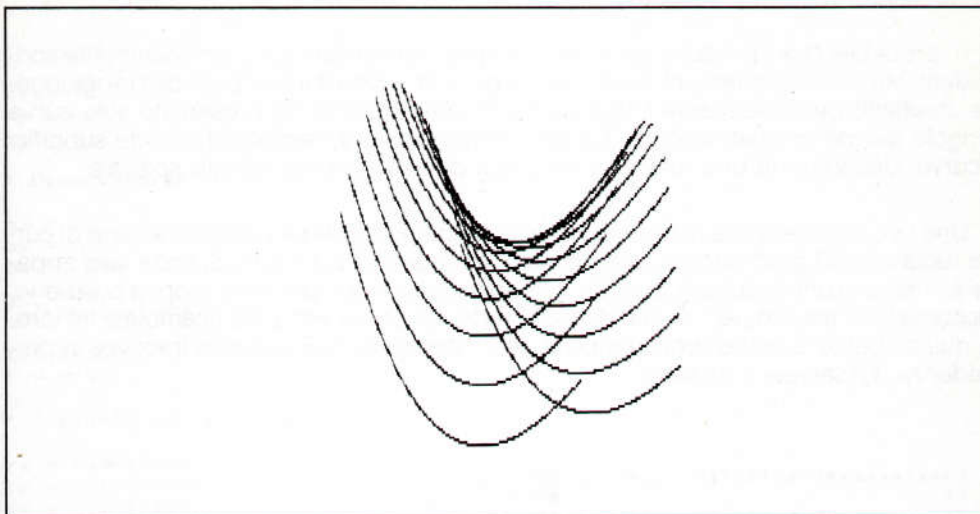


Fig. 5.1 Funzioni tridimensionali 1

Dopo l'apertura di un nuovo schermo con finestra, il programma esegue alcune inizializzazioni, che riguardano prima di tutto i diversi parametri per le trasformazioni. La routine dovrà essere in grado di far ruotare il grafico della funzione attorno ai tre assi spaziali, utilizzando la matrice di rotazione totale, che abbiamo visto nell'ultimo Capitolo, per le rotazioni attorno agli assi x, y e z. Di conseguenza, calcoleremo prima di tutto i fattori di rotazione A, B, C, D, E, F, G, H e I, che rimarranno costanti nell'intero programma (ci risparmieremo con ciò calcoli inutili nei loop successivi).

In questi parametri di trasformazione abbiamo naturalmente spazio a disposizione per modifiche. I valori momentaneamente impostati producono una grafica veramente meravigliosa.

A questo punto, il programma imposta dei valori di inizio e di termine nonché le ampiezze di passo per le coordinate x e z, che verranno incrementate (o decrementate) nei loop FOR...NEXT seguenti. Nel caso in cui si desideri ottenere una visione grossolana dell'immagine, si raccomanda di elevare da 10 a 30 ste.x (cioè l'ampiezza di x), poiché l'esecuzione del disegno richiede normalmente un po' di tempo.

La struttura di entrambi i loop FOR...NEXT nidificati l'uno nell'altro è già stata studiata. A questo punto i valori corrispondenti della funzione vengono calcolati, ampliati, trasferiti, ruotati e proiettati tramite proiezione centrale, quindi con le coordinate risultanti, viene tracciato un punto. Si dovrà attendere un po' di tempo prima che il risultato venga visualizzato. E' sufficiente una pressione di tasto per porre fine al programma.

5.1.2 Quadrettatura (Crosshatching)

E' probabile che il prodotto del nostro programma non ci abbia particolarmente soddisfatti. Ma lasciatemi tempo! Grazie ad un piccolo trucco è infatti possibile raggiungere un effetto particolarmente interessante. Finora abbiamo rappresentato solo curve singole per determinati valori z. Tuttavia, sappiamo che esistono stupende superfici ricurve, decorate da una rete, che amplifica particolarmente l'effetto spaziale.

Una tale quadrettatura (Crosshatching) si raggiunge disegnando di una serie di curve ruotate di 90 gradi attorno all'asse y sull'immagine di cui sopra. Si tratta solo apparentemente di una rotazione, poiché l'immagine rimane sempre nella propria posizione, siccome le ampiezze per i comandi di STEP dei loop z e x vengono scambiate fra loro. A questo punto le curve sono completamente perpendicolari a quelle tracciate in precedenza. Osservate il risultato!

```
' ********************************
' **                            **
' **  Funzioni tridimensionali 2 **
' **                            **
' ********************************

PI = 3.141593

SCREEN 2,640,200,2,2              ' Nuovo schermo
WINDOW 4,,(0,0)-(631,186),0,2 ' Nuova finestra

PALETTE 0,  0,  0,  0             ' Impostazione colori
PALETTE 1, .8,  0,.93
PALETTE 2,.47,.87,  1
PALETTE 3,  1, .6,.67

COLOR 2

' Parametri di transformazione:
' Scala:

sx = 40
sy = 6
sz = 20

' Translazione:

tx = 0
ty = 0
tz = 0
```

```
' Rotazione:

rx = 15
ry = -45
rz = 0

' Osservatore:

bx = 0
by = 0
bz = -300

' Translazione piano:

tex = 300
tey = 90

' Costanti per le rotazioni:

rx = rx*PI/180
ry = ry*PI/180
rz = rz*PI/180

si.x = SIN(rx) : co.x = COS(rx)
si.y = SIN(ry) : co.y = COS(ry)
si.z = SIN(rz) : co.z = COS(rz)

A = co.y * co.z
B = co.y * si.z
C = -si.y
D = si.x*si.y*co.z - co.x*si.z
E = si.x*si.y*si.z + co.x*co.z
F = si.x*co.y
G = co.x*si.y*co.z + si.x*si.z
H = co.x*si.y*si.z - si.x*co.z
I = co.x*co.y

' Valori di inizio/fine/ampiezza passi:

sta.x = 3 : en.x = -3 : ste.x = -.01
sta.z = 4 : en.z = -3 : ste.z = -.5

' Funzione:

DEF FNfun(x,z) = x*x - z*z


' Disegno:
```

```
FOR za = 1 TO 2

  FOR z=sta.z TO en.z STEP ste.z
    FOR x=sta.x TO en.x STEP ste.x

      ' Calcolo valore della funzione:

      y = FNfun(x,z)

      ' Trasformazioni:

      xt1 = sx*x + tx              ' Scalatura
      yt1 = sy*y + ty              ' e translazione
      zt1 = sz*z + tz

      xt2 = xt1*A + yt1*B + zt1*C ' Rotazione
      yt2 = xt1*D + yt1*E + zt1*F
      zt2 = xt1*G + yt1*H + zt1*I

      ' Proiezione centrale:

      zwis = zt2 - bz
      xe = bx - bz * (xt2-bx)/zwis
      ye = by - bz * (yt2-by)/zwis

      ' Tracciamento del punto:

      PSET (tex + xe, tey - ye)

    NEXT x
  NEXT z

  IF za = 1 THEN         ' Al primo passaggio le ampiezze
    zwis = ste.x         ' dei passi vengono scambiate
    ste.x = ste.z
    ste.z = zwis

    COLOR 3
  END IF

NEXT za

WHILE (INKEY$ = "")
WEND

WINDOW CLOSE 4
SCREEN CLOSE 2
```

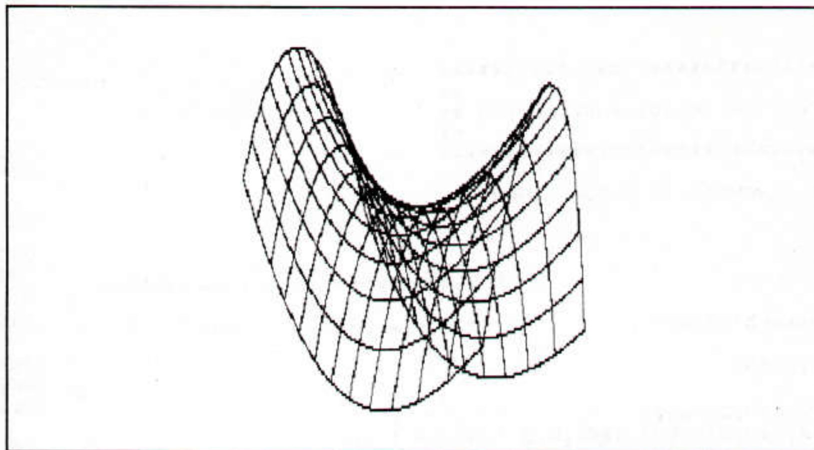


Fig. 5.2 Funzioni tridimensionali 2

Il programma non è cambiato molto. Si è aggiunto semplicemente un terzo loop FOR...NEXT che contiene tutto il resto. Il grafico viene in questo modo tracciato due volte su due assi diversi. Se il risultato non fosse ancora soddisfacente, si consiglia di leggere più avanti.

5.1.3 Le linee nascoste

Con le nostre funzioni non abbiamo fatto molti progressi rispetto al punto in cui eravamo nell'ultimo Capitolo con gli oggetti in generale, il grafico tridimensionale è ancora trasparente e le linee coperte sono ancora visibili. Osserveremo ora un procedimento semplice per l'eliminazione di questo difetto.

Nella vita di tutti i giorni normalmente non riusciamo a guardare attraverso le cose che osserviamo a meno che non siano di vetro, mentre qui accade il contrario. Le nostre linee si intersecano, mentre quelle anteriori, che sorreggono una superficie, dovrebbero coprire quelle posteriori.

Cerchiamo di prendere vantaggio dal fatto che iniziamo a disegnare con valori elevati di z (ecc.) e il grafico viene eseguito dalla parte posteriore verso la parte anteriore. Consideriamo ogni valore z come una superficie parallela ai piani x-y. Quando tracciamo tali superfici dalla parte posteriore a quella anteriore, quelle anteriori copriranno sempre più quelle posteriori. Nel programma cancelleremo quindi intere superfici, in modo che tutte le superfici retrostanti vengano eliminate. Cancelleremo quindi dal punto appena calcolato e tracciato della curva, in linea verticale fino al bordo inferiore della finestra.

In questo modo otteniamo un controllo della struttura grafica della funzione. Se nel suddetto programma aggiungiamo un paio di righe, verremo ad avere chiaramente il seguente effetto:

```
' ********************************
' **                            **
' **  Funzioni tridimensionali 3 **
' **                            **
' ********************************

IF FRE(-1)+FRE(0) < 36000& THEN
  END
END IF

CLEAR ,36000&                    ' 36000 Bytes per l' AmigaBASIC

DIM fenster%(14962)              ' Memoria immagine video

PI = 3.141593

SCREEN 2,640,200,2,2             ' Nuovo schermo
WINDOW 4,,(0,0)-(631,186),0,2 ' Nuova finestra

PALETTE 0,  0,  0,  0            ' Impostazione colori
PALETTE 1, .8,  0,.93
PALETTE 2,.47,.87,  1
PALETTE 3,  1, .6,.67

COLOR 2

' Parametri di trasformazione:
' Scala:

sx = 40
sy = 6
sz = 20

' Translazione:

tx = 0
ty = 0
tz = 0

' Rotazione:

rx = 15
ry = -45
rz = 0
```

```
' Osservatore:

bx = 0
by = 0
bz = -300

' Translazione piano:

tex = 300
tey = 90

' Costanti per le rotazioni:

rx = rx*PI/180
ry = ry*PI/180
rz = rz*PI/180

si.x = SIN(rx) : co.x = COS(rx)
si.y = SIN(ry) : co.y = COS(ry)
si.z = SIN(rz) : co.z = COS(rz)

A = co.y * co.z
B = co.y * si.z
C = -si.y
D = si.x*si.y*co.z - co.x*si.z
E = si.x*si.y*si.z + co.x*co.z
F = si.x*co.y
G = co.x*si.y*co.z + si.x*si.z
H = co.x*si.y*si.z - si.x*co.z
I = co.x*co.y

' Valori di inizio/fine/ampiezza passi:

sta.x = 3 : en.x = -3 : ste.x = -.01
sta.z = 4 : en.z = -3 : ste.z = -.5

' Funzione:
DEF FNfun(x,z) = x*x - z*z

' Disegno:

FOR za = 1 TO 2

  FOR z=sta.z TO en.z STEP ste.z
    FOR x=sta.x TO en.x STEP ste.x

      ' Calcolo valore della funzione:

      y = FNfun(x,z)
```

```
      ' Trasformazioni:

      xt1 = sx*x + tx              ' Scalatura
      yt1 = sy*y + ty              ' e translazione
      zt1 = sz*z + tz

      xt2 = xt1*A + yt1*B + zt1*C ' Rotazione
      yt2 = xt1*D + yt1*E + zt1*F
      zt2 = xt1*G + yt1*H + zt1*I

      ' Proiezione centrale:

      zwis = zt2 - bz
      xe = bx - bz * (xt2-bx)/zwis
      ye = by - bz * (yt2-by)/zwis

      '  Disegno del punto e cancellazione linea sotto di esso:

      PSET (tex + xe, tey - ye)
      LINE (tex+xe,tey-ye+1)-(tex+xe,200),0

    NEXT x
  NEXT z

  IF za = 1 THEN        ' Al primo passaggio le ampiezze
    zwis = ste.x        ' dei passi sono scambiate
    ste.x = ste.z
    ste.z = zwis

    ' Memorizzazione intermedia finestra:

    GET (0,0)-(631,186),fenster%
    COLOR 3
    CLS                 ' e cancellazione
  END IF

NEXT za

PUT (0,0),fenster%,OR ' Operazione di OR con la seconda finestra

WHILE (INKEY$ = "")
WEND

WINDOW CLOSE 4
SCREEN CLOSE 2
```

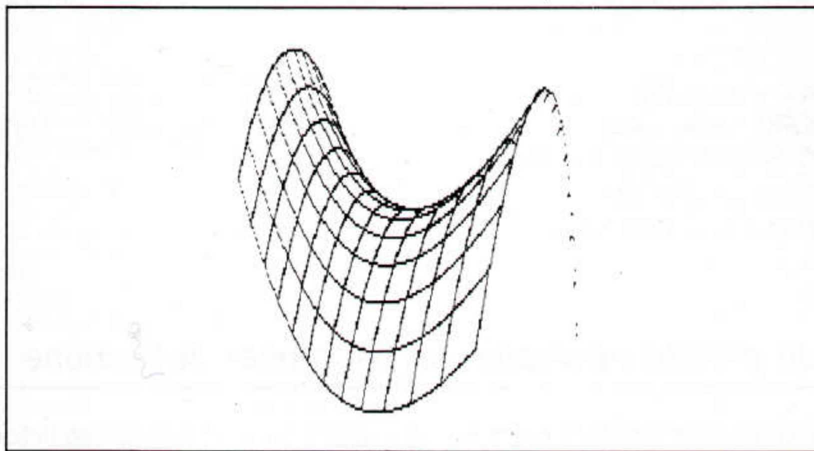


Fig. 5.3 Funzioni tridimensionali 3

Cosa accade? In questo momento nel nostro programma le due parti di immagine non vengono disegnate l'una sull'altra. Infatti la routine effettua una memorizzazione intermedia della prima parte. Per questo serve il comando GET. Esso trasmette il contenuto totale della finestra alla matrice integrale finestra %( ), che è stata scelta all'inizio del programma (ved. manuale AmigaBasic sotto la parola GET). A tale scopo si dovrebbe aver riservato una memoria sufficiente (CLEAR).

La prima parte effettua una cancellazione dei punti sotto ogni punto con il comando LINE, mentre il risultato viene memorizzato in maniera intermedia per mezzo di GET. Lo schermo ridiventa vuoto e la seconda parte del grafico viene creata nella stessa maniera. Quando alla fine è tutto pronto, si azionerà il comando PUT. Esso, tramite un'operazione logica di OR, fonde i due grafici in uno solo.

Ciò che non è ancora soddisfacente al cento percento sono i margini. Si potrebbe anche desiderare di vedere la curva dalla parte inferiore. Ciò non è permesso dall'algoritmo. Per eliminare tale manchevolezza, si potrebbe cancellare la linea successiva, ma non all'interno del bordo inferiore dell'immagine, bensì solo dal punto sottostante. E anche questo problema è risolto. Il disegno effettuerà più volte delle memorizzazioni intermedie tramite tale metodo, per una durata notevole. Tentate pure qualche volta a trasformare questo programma!

Ecco un'idea per ottimizzare il procedimento di cancellazione: annotare di volta in volta le coordinate, finora disegnate, dei piani y momentaneamente inferiori e cancellare solo fino a quel punto.

Rappresenteremo nella prossima sezione un altro interessante algoritmo per la cancellazione delle linee nascoste nelle funzioni tridimensionali.

Eccovi ora qualche funzione che potrete visualizzare con il vostro programma:

$y = x^2 + z^2$
$y = 1/(1 + x^2 + z^2)$
$y = SQR(1 - x^2/4 - z^2/9)$
$y = 20*SIN(0.1*SQR(x^2 + z^2)$
$y = SIN(x)/x + SIN(z)/z$
$y = SIN(1/x)/x + SIN(1/z)/x$

5.2 Un piccolo intervallo: un bel plotter di funzione

Ci troviamo ora in grado di rappresentare sullo schermo meravigliose funzioni tridimensionali. Ciò che ci manca è un programma universale, che sia adatto ad ogni funzione a piacere e che raffini un pochino le rappresentazioni. Di conseguenza, bisogna occuparsi del programma seguente, come ulteriore tappa importante di questo libro.

Iniziamo un'altra volta e lasciamoci affascinare dalla singolare bellezza della grafica dell'Amiga:

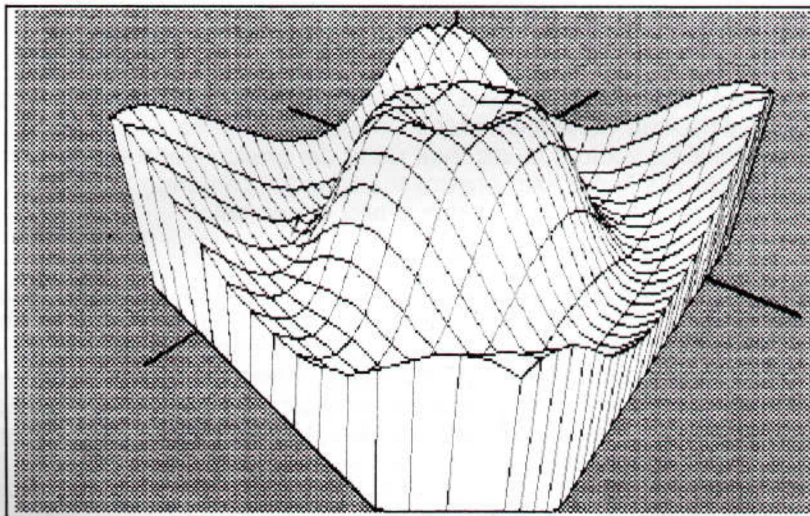


Fig. 5.4 Plotter di funzioni tridimensionali 1

```
'***************************
'**       Il grande        **
'**   Plotter funzionale   **
'**    tridimensionale     **
'**                        **
'***************************

'Apertura di Schermi e Finestre / Impostazione dei colori:
SCREEN 2,640,200,3,2
wx% = 631 '
wy% = 186+10
WINDOW 4,,(0,0)-(wx%,wy%-10),0,2

PALETTE 0,  0,  0,  0
PALETTE 1, .8,  0,.93 'Colore Motivo di sfondo
PALETTE 2,.47,.87,  1 'Colore rete
PALETTE 3,  1,.47,.53 'Colore superiore superfice rete
PALETTE 4, .4, .6,  1 'Colore sistema delle coordinate
PALETTE 5,  1, .6,.67 'Colore cornice anteriore
PALETTE 6, .8,.33, .4 'Colore cornice laterale

COLOR 2


'Il Flag seguente indica se vogliamo immettere
'i diversi Parametri tramite Input (flag%=0)
'o direttamente nel programma (flag%=1):

flag% = 1


'Qui immettiamo la funzione tridimensionale
'desiderata f(x,z):

 DEF FNfun(x,z) = 20*SIN(.1*SQR(x*x+z*z))
'********************************************


'Immissioni di Input:
'*******************

LOCATE  1,15 : PRINT "Immissione dei parametri di funzione"
LOCATE  2,15 : PRINT "************************************"
LOCATE  4,5  : PRINT "Intervallo x:      Partenza:"
LOCATE  5,24 : PRINT "Fine:"
LOCATE  6,5  : PRINT "Intervallo y:      Partenza:"
LOCATE  7,24 : PRINT "Fine:"
LOCATE  8,5  : PRINT "Intervallo z:      Partenza:"
LOCATE  9,24 : PRINT "Fine:"
LOCATE 11,5  : PRINT "Messa in scala dello spazio: Fattore x:"
LOCATE 12,23 : PRINT "Fattore y:"
LOCATE 13,23 : PRINT "Fattore z:"
LOCATE 15,5  : PRINT "Rotazione spaziale: Asse x:"
LOCATE 16,23 : PRINT "Asse y:"
```

```
LOCATE 17,23 : PRINT "Asse z:"
LOCATE 19,5  : PRINT "Numero dei punti x calcolati:"
LOCATE 20,5  : PRINT "Grandezza rete direzione x:"
LOCATE 21,5  : PRINT "e in direzione z:"
LOCATE 22,5  : PRINT "Cornice              (s/n):"
LOCATE 23,5  : PRINT "Sistema coordinate (s/n):"

IF flag% = 1 THEN
  GOTO start      'Parametri dal programma
END IF

'Input:
LOCATE  4,35 : INPUT xa
LOCATE  5,35 : INPUT xb
LOCATE  6,35 : INPUT ya
LOCATE  7,35 : INPUT yb
LOCATE  8,35 : INPUT za
LOCATE  9,35 : INPUT zb
LOCATE 11,35 : INPUT sx
LOCATE 12,35 : INPUT sy
LOCATE 13,35 : INPUT sz
LOCATE 15,35 : INPUT rx
LOCATE 16,35 : INPUT ry
LOCATE 17,35 : INPUT rz
LOCATE 19,35 : INPUT rech.gen
LOCATE 20,35 : INPUT netz.x
LOCATE 21,35 : INPUT netz.z
LOCATE 22,35 : INPUT umrahm$
LOCATE 23,35 : INPUT kreuz$

'alcuni altri valori da impostare:
tx=0 : ty=0 : tz=0
beo.x = 0 : beo.y = 0 : beo.z = -400

'Parametri di trasformazione del piano:
tex& = wx%/2      'circa il centro della finestra
tey& = wy%/2.5
sex& = 1
sey& = 2

GOTO weiter


'In alternativa potremo immettere qui i diversi
'Parametri, anche direttamente nel programma:
start:
xa = -60          'Partenza Intervallo x
xb =  60          'Fine Intervallo x
ya = -30          'Partenza Intervallo y
yb =  60          'Fine Intervallo y
za = -60          'Partenza Intervallo z
zb =  60          'Fine Intervallo z
sx = 3            'Scala dello spazio in direzione x
```

```
sy = 3              'Scala dello spazio in direzione y
sz = 3              'Scala dello spazio in direzione z
tx = 0              'Traslazione dello spazio in direzione x
ty = 0              'Traslazione dello spazio in direzione y
tz = 0              'Traslazione dello spazio in direzione z
rx = 30             'Rotazione dello spazio attorno all'asse x
ry = -40            'Rotazione dello spazio attorno all'asse y
rz = 0              'Rotazione dello spazio attorno all'asse z
beo.x = 0           'Coordinate osservatore
beo.y = 0
beo.z = -400
rech.gen = 30       'Numero dei punti da calcolare
                    'per ogni intervallo x (Riga)
netz.x = 15         'Dimensione reticolo in numero di righe per
                    'Intervallo x (Righe)
                    '(rech.gen deve essere un multiplo intero
                    'di netz.x)
netz.z = 30         'Dimensione reticolo in numero di righe per
                    'Intervallo z
umrahm$ = "s"       'Attivazione cornice
kreuz$ = "s"        'Attivazione sistema coordinate

'Parametri di trasformazione del piano:---
tex& = wx%/1.8  'circa il centro della finestra
tey& = wy%/3
sex& = 1            'Messa in scala
sey& = 2


weiter:

PI = 3.141593

' Matrici di punti per la memorizzazione intermedia di due colonne
DIM merk.x&(rech.gen-1,1), merk.y&(rech.gen-1,1)

'Memorizzazione del  Pattern:
DIM feld1%(3), feld2%(3)

' Profondita' suolo:
boden = ya

' Costanti per la rotazione:
rx = rx*PI/180                  'Gradi -> Radianti
ry = ry*PI/180
rz = ry*PI/180

A = COS(ry)*COS(rz)
B = COS(ry)*SIN(rz)
C = -SIN(ry)
D = SIN(rx)*SIN(ry)*COS(rz) - COS(rx)*SIN(rz)
E = SIN(rx)*SIN(ry)*SIN(rz) + COS(rx)*COS(rz)
F = SIN(rx)*COS(ry)
```

```
G = COS(rx)*SIN(ry)*COS(rz) + SIN(rx)*SIN(rz)
H = COS(rx)*SIN(ry)*SIN(rz) - SIN(rx)*COS(rz)
I = COS(rx)*COS(ry)

'Calcolo ampiezza:
IF rech.gen < 2 THEN
  PRINT "Precisione di calcolo troppo bassa!" : GOTO Stp
END IF
ste.z = (zb-za)/(netz.z-1)
ste.x = (xb-xa)/(rech.gen-1)
IF ste.z<0 OR ste.x<0 OR ya>yb THEN
  PRINT "Errore nell'input di intervallo!"  : GOTO Stp
END IF

' Numero passaggi di calcolo per ogni riga verticale di rete:
anz.netz% = INT(rech.gen/netz.x)

' Pattern per la definizione dell'intero schermo:
feld1%(0) = &HCCCC
feld1%(1) = &H3333
feld1%(2) = &HCCCC
feld1%(3) = &H3333

' e per la funzione:
feld2%(0) = &HFFFF      '&HAAAA
feld2%(1) = &HFFFF      '&H5555
feld2%(2) = &HFFFF      '&HAAAA
feld2%(3) = &HFFFF      '&H5555

PATTERN ,feld1%
LINE (0,0)-(wx%,wy%),1,BF     ' Riempimento schermo con il motivo

PATTERN ,feld2%               ' Impostazione secondo motivo

POKE WINDOW(8)+27,2           ' Colore dell'Area-OUTLINE-Pen
flags = WINDOW(8)+32
POKEW flags,PEEK(flags) OR 8  ' Impostazione dell'AreaOutline-Flag

r.akt% = 0                    ' Numero della matrice di righe attuale
r.last% = 1                   ' Numero dell'ultima matrice di righe

' Routine di tracciatura:
```

```
' Tracciatura del sistema di coordinate:
IF kreuz$ = "s" THEN
  IF za<0 AND zb>0 AND xa<0 AND xb>0 AND ya<0 AND yb>0 THEN
    COLOR 4
    x=xa*1.3:y=0:z=0:GOSUB transform:PSET (x1&,y1&)
    x=xb*1.3:y=0:z=0:GOSUB transform:LINE -(x1&,y1&)
    x=0:y=ya*1.3:z=0:GOSUB transform:PSET (x1&,y1&)
    x=0:y=yb*1.3:z=0:GOSUB transform:LINE -(x1&,y1&)
    x=0:y=0:z=za*1.3:GOSUB transform:PSET (x1&,y1&)
    x=0:y=0:z=zb*1.3:GOSUB transform:LINE -(x1&,y1&)
  END IF
END IF

FOR z=zb TO za STEP -ste.z

  zwis% = r.akt%              ' Scambio numeri della matrice di righe
  r.akt% = r.last%
  r.last% = zwis%
  n.zaehl% = -1               ' Contatore per reticolo
  r.zaehl% = 0                ' Contatore per matrice di righe
  COLOR 3

  FOR x=xa TO xb STEP ste.x

    'Calcolo valore funzione:
    y = FNfun(x,z)

    GOSUB transform          ' Trasformazione coordinate
    GOSUB zeichne            ' tracciato campo / annotazione coord. ecc.

  NEXT x

  IF z<>zb AND umrahm$="s" THEN

    rette = z                ' salvare z
    rette5 = x
    COLOR 6

    ' Tracciato parete destra/sinistra:

    FOR seite% = 1 TO 2

      IF seite% = 1 THEN     ' Parete sinistra
        ' Visibile?
        IF merk.x&(0,r.last%) >= merk.x&(0,r.akt%) THEN skip
        xvu = xa : zvu = z : yvu = boden
        xhu = xa : zhu = z+ste.z : yhu = boden
        xho = xa : zho = zhu : yho = FNfun(xho,zho)
        xvo = xa : zvo = zvu : yvo = FNfun(xvo,zvo)
      ELSE                   ' Parete destra
        ' Visibile?
        IF merk.x&(r.zaehl%-1,r.last%) <= merk.x&(r.zaehl%-1,r.akt%) THEN skip
        xvu = x-ste.x : zvu = z : yvu = boden
        xhu = xvu : zhu = z+ste.z : yhu = boden
```

```
        xho = xvu : zho = zhu : yho = FNfun(xho,zho)
        xvo = xvu : zvo = zvu : yvo = FNfun(xvo,zvo)
      END IF

       ' Punti anteriori inferiori:
       x = xvu : y = yvu : z = zvu
       GOSUB transform
       IF  y1& >= wy%  THEN  y1& = wy%-1
       hold.x& = x1& : hold.y& = y1&
       GOSUB makearea
       ' Punti posteriori inferiori:
       x = xhu : y = yhu : z = zhu
       GOSUB transform
       IF  y1& >= wy%  THEN  y1& = wy%-1
       GOSUB makearea
       ' Punti superiori posteriori:
       x = xho : y = yho : z = zho
       GOSUB transform
       GOSUB makearea
       ' Punti superiori anteriori:
       x = xvo : y = yvo : z = zvo
       GOSUB transform
       GOSUB makearea

       AREAFILL 0                    ' riempimento

       x = rette5
       z = rette
     skip:
     NEXT seite%
     z = rette
  END IF

NEXT z

' Tracciato parete anteriore:
IF umrahm$ = "s" THEN

  zwis% = r.akt%               ' Scambio numeri matrici righe
  r.akt% = r.last%
  r.last% = zwis%
  n.zaehl% = -1                ' Contatore per reticolo
  r.zaehl% = 0                 ' Contatore per matrici righe
  COLOR 5

  FOR x=xa TO xb STEP ste.x

    y = boden

    GOSUB transform          'Trasformazione coordinate
    IF x+ste.x >= xb THEN x1&=hold.x& : y1&=hold.y&
    IF y1& >= wy% THEN y1& = wy%-1
    GOSUB zeichne             ' tracciato campo / annotazione coord. ecc.
```

```
  NEXT x
 END IF

 Stp:
 WHILE (INKEY$="")            ' Attesa tasto
 WEND

 WINDOW CLOSE 4
 SCREEN CLOSE 2

 END

 ' Annotazione delle coordinate nella matrice ed
 ' eventualmente tracciato superficie
zeichne:
   'Memorizzazione nella matrice:
   merk.x&(r.zaehl%, r.akt%) = x1&
   merk.y&(r.zaehl%, r.akt%) = y1&

   n.zaehl% = n.zaehl% + 1
   'Tracciato di un riquadro della rete:
   IF n.zaehl%=anz.netz% AND z<>zb AND y>=ya AND y<=yb THEN
     n.zaehl% = 0               ' Reset del contatore

     'Inizializzazione del polinomio di Area:
     FOR ind = r.zaehl%-anz.netz% TO r.zaehl%
       x1& = merk.x&(ind,r.akt%)
       y1& = merk.y&(ind,r.akt%)
       GOSUB makearea
     NEXT ind
     FOR ind = r.zaehl% TO r.zaehl%-anz.netz% STEP -1
       x1& = merk.x&(ind,r.last%)
       y1& = merk.y&(ind,r.last%)
       GOSUB makearea
     NEXT ind

     AREAFILL 0                 'Riempimento del polinomio

  END IF
  r.zaehl% = r.zaehl% + 1   ' punto successivo
RETURN


' Routine di trasformazione:
' che trasforma le coord. tridim. espresse in x, y, z
' in coord. bidim. espresse in x1&, y1&

transform:
  'Esecuzione delle Trasformazioni:
  xt1 = sx*x + tx           ' Messa in scala e traslazione
  yt1 = sy*y + ty
  zt1 = sz*z + tz
```

```
  xt2 = xt1*A + yt1*B + zt1*C ' Rotazioni
  yt2 = xt1*D + yt1*E + zt1*F
  zt2 = xt1*G + yt1*H + zt1*I

  'Proiezione centrale:
  zwis = zt2 - beo.z
  x1& = beo.x - beo.z * (xt2-beo.x)/zwis
  y1& = beo.y - beo.z * (yt2-beo.y)/zwis

  'Traslazione del piano:
  x1& = tex& + x1&/sex&
  y1& = tey& - y1&/sey&
RETURN

'Mini-Clipping e chiamata dell' Area con x1&, y1&:
makearea:
  ' Mini-Clipping:
  IF x1&>=0 AND x1&<wx% AND y1&>=0 AND y1&<wy% THEN
    AREA(x1&,y1&)
  END IF
RETURN
```

Incredibile! Questo programma disegna, nel modo che stiamo vedendo, una funzione ben determinata in una maniera ben determinata. Tutto ciò, ed altro, viene indicato nella variabile flag %. In questo caso essa è stata impostata a 1. Questo flag indica chiaramente se vogliamo inserire i diversi parametri modificabili nel programma per mezzo di INPUT dopo ogni inizio di programma (flag% = 0), oppure se devono venire utilizzati direttamente i parametri che si trovano nel programma (flag% = 1). Il flag è stato introdotto al fine di risparmiare il controllo delle funzioni con i parametri. Sarebbe particolarmente complicato dovere inserire di nuovo tutti i parametri ad ogni svolgimento dei test, quando si desidera cambiarne uno solo.

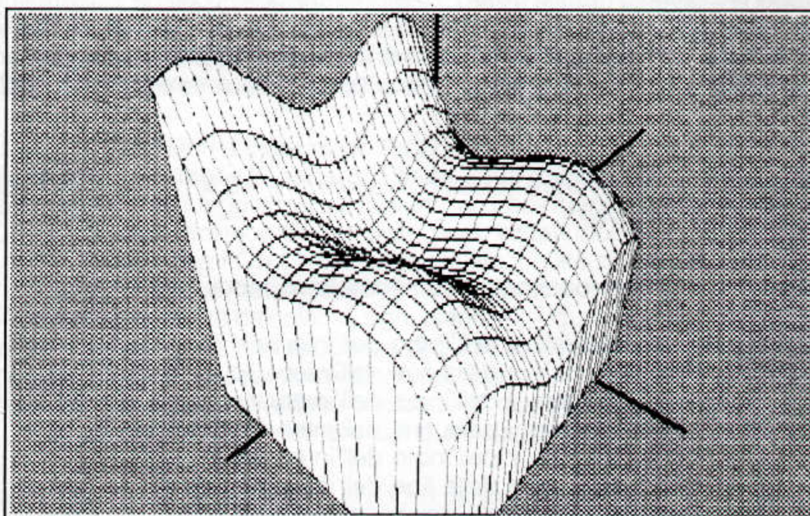


Fig. 5.5 Plotter di funzioni tridimensionali 2

Il comando DEF serve per la definizione della funzione tridimensionale che si desidera tracciare. Fate attenzione al fatto che non tutte le funzioni sono adatte a venire rappresentate sullo schermo. A causa di funzioni con numerose lacune di definizione possono emergere innumerevoli problemi. Per es., se durante il calcolo il nominatore diventa uguale a zero, emerge naturalmente il messaggio di errore DIVISION BY ZERO. Radici negative o valori troppo grandi, portano all'arresto del programma. Bisognerà quindi scegliere con criterio gli intervalli dei valori!

Se si desidera rappresentare tali funzioni in posizioni indefinite, ecco un trucco: installare una routine ON ERROR (Attenzione! La funzione verrà calcolata in numerosi punti del programma), o utilizzare le seguenti caratteristiche dal Basic dell'Amiga. Immettere in modo diretto:

```
PRINT 4=4;"/";4=2
```

Risultato: -1 / 0

Con ciò abbiamo appena ottenuto il risultato di un confronto logico. Ad ogni operazione logica l'AmigaBasic attribuisce un valore, con il quale è possibile anche calcolare. Se l'espressione logica è vera, si ottiene il valore -1, se questa è falsa, il risultato è 0. Tale confronto (anche con ">", "<", "<>", ecc.) potrà venire inserito naturalmente nelle nostre funzioni.

La funzione f(x) = sin(x)/x per es. non viene definita nella posizione x = 0. Affinché però il programma possa calcolare anche questa posizione, scriviamo:

f(x) = sin(x)/(x-(x=0))

Non appena x diventa uguale a zero, l'espressione (x = 0) ottiene il valore -1, il quale deve venire sotratto da x (quindi da 0). Il risultato per x = 0 è: f(0) = sin(0)/(0-(-1)). Il nominatore diventa uguale a 1. Il punto che ne risulta non rappresenta certamente nessun punto valido della curva, ma ciò non si nota particolarmente nelle applicazioni pratiche.

Con le radici, si dovrà lavorare invece con la funzione ABS( ): f(x) = SQR(ABS(x)) non fornisce mai valori non validi.

Arriviamo ora ai diversi parametri che si dovranno modificare:

xa	Coordinata spaziale x dell'inizio dell'intervallo x
xb	Coordinata spaziale x della fine dell'intervallo x
ya	Coordinata spaziale y dell'inizio dell'intervallo y
yb	Coordinata spaziale y della fine dell'intervallo y
za	Coordinata spaziale z dell'inizio dell'intervallo z
zb	Coordinata spaziale z della fine dell'intervallo z
sx	Scala dello spazio della funzione in direzione x
sy	Scala dello spazio della funzione in direzione y
sz	Scala dello spazio della funzione in direzione z
tx	Traslazione dello spazio in direzione x
ty	Traslazione dello spazio in direzione y
tz	Traslazione dello spazio in direzione z
rx	Rotazione dello spazio attorno all'asse x
ry	Rotazione dello spazio attorno all'asse y
rz	Rotazione dello spazio attorno all'asse z
beo.x	Coordinata x dell'osservatore
beo.y	Coordinata y dell'osservatore
beo.z	Coordinata z dell'osservatore
rech.gen	Numero dei punti da calcolare per intervallo x (righe)
netz.x	Densità della rete x (numero delle linee di rete per intervallo x; rech.gen deve essere un multiplo intero di netz.x!)
netz.z	Densità della rete z (numero delle linee di rete per intervallo z; contemporaneamente numero dei punti da calcolare per intervallo z)
umrahm$	Flag per incorniciatura inserita/disinserita
kreuz$	Flag per sistema di coordinate inserito/disinserito
tex&	Traslazione dei piani in direzione x (posizione x del grafico nella finestra)
tey&	Traslazione dei piani in direzione y (posizione y del grafico nella finestra)
sex&	Scala dei piani in direzione x (coordinata x per 1/sex&)
sey&	Scala dei piani in direzione y (coordinata y per 1/sey&)

Per motivi di spazio non è possibile inserirli tutti tramite INPUT. La cosa migliore è di provare un po'. In questa maniera potremo verificare le conseguenze di ogni singolo parametro. D'altra parte incontreremo cose che abbiamo già imparato (per es. i parametri di trasformazione). L'importante è introdurre i parametri di intervallo in modo che valga: ya<yb,xa<xb e za<zb. yb lo dovremo scegliere il più grande possibile. Eviteremo angoli maggiori di 45 gradi o minori di -45 gradi (meglio scegliere valori vicini a zero).

La routine apre, all'inizio del programma, un nuovo schermo completo con una finestra interna. Lo schermo è dotato di tre bit-plane. Sono disponibili anche otto colori diversi. Nel nostro programma ne utilizzeremo solo sette, che verranno definiti tramite **PALETTE**. Poi, vediamo cosa succede con l'inserimento dei parametri.

Infatti, il vero programma inizia dopo l'etichetta di salto "avanti:". Qui vengono inizializzate alcune cose molto importanti. Per es., due matrici (**merk.x&(,)** e **merk.y&(,)**) che ci accompagnano lungo l'intero programma. Esse contengono le coordinate intere di due serie z della funzione da calcolare (ved. sotto). Inoltre, il programma calcola i valori di STEP per i loop FOR...NEXT ulteriori (ste.x e ste.z) che derivano dagli intervalli x e z e dalla precisione matematica indicata. La precisione matematica indica quanti punti singoli devono venire calcolati in ogni piano y-z, i quali in seguito verranno collegati con linee per produrre una sequenza di curve in rapporto l'una con l'altra. La variabile netz.z contiene la stessa informazione per i piani x e y. Invece netz.x indica quante linee di rete devono incrociare le linee orizzontali di rete per ogni intervallo x. **anz.netz%** determina, di conseguenza, il numero dei punti da calcolare tra le due linee di rete.

Segue poi il sistema di coordinate: sei punti vengono calcolati, trasformati e proiettati (tramite il sottoprogramma "transform") e vengono tracciate tre linee.

Conosciamo già entrambi i loop FOR...NEXT nidificati l'uno nell'altro, che si trovavano nei programmi descritti nei Capitoli precedenti. Qui essi sonó naturalmente più estesi, ma il principio è sempre lo stesso.

Ora dovremo prendere in considerazione come risolvere il problema delle linee nascoste in questo programma, cioè in modo diverso da quello che abbiamo fin qui imparato.

Il programma non traccia immediatamente sullo schermo tutti i punti calcolati della funzione. I valori x e y di ogni punto di una sequenza x vengono messi nelle matrici sopra menzionate **merk.x&(n,r)** e **merk.y&(n,r)**. n indica la posizione del punto all'interno di una sequenza. r rappresenta 0 o 1. Due sequenze x impostate l'una dopo l'altra vengono sempre memorizzate nelle matrici. Questa operazione ha un obiettivo ben preciso. Da ogni punto non vengono tracciate linee semplici, ma i punti in direzione x e anche in direzione z vengono collegati l'uno con l'altro tramite linee (quadrettatura). La superficie, delimitata da queste linee (sup. a rete) riempie il programma con un colore. In conseguenza essa ridipinge tutto ciò che era stato tracciato dietro. L'immagine risulterà quindi composta quasi come un mosaico. Potrebbero emergere problemi, ma solo in caso di grandi rotazioni rx, ry ed rz in alcune funzioni.

Dopo che è stato calcolato nel loop x anche un valore di funzione, tale punto viene trasformato e proiettato nel sottoprogramma "transform" alla vecchia maniera. Le coordinate dei piani già approntate si trovano in x1& e y1& e vengono trasmesse al sottoprogramma "zeichne", che si trova in entrambe le matrici sopracitate. E' qui che viene deciso se tracciare veramente: al primo passaggio dei loop z (z=zb) non è ancora piena nessuna matrice. Generalmente, è possibile tracciare solo dopo il secondo passaggio del loop z. Nel caso in cui debbano venire calcolati ulteriori punti per ogni linea di rete verticale, il disegno dovrà restare sospeso finché non si sarà ottenuta una linea di rete.

Tracceremo una superficie. A tale scopo utilizzeremo il comando **AREA** oppure **AREAFILL**, che riempirà un poligono a piacere. Il programma trasmetterà ogni punto angolare della superficie, ottenuto dalla matrice merk.x&(,) e merk.y(,), al comando AREA. Questi punti sono perlomeno quattro, ma potranno essere anche sei, otto, dieci ecc. a seconda dell'esattezza di calcolo e del numero delle linee di rete.

Quando una linea è stata elaborata, a seconda del Flag umrahm$ e della prospettiva, potrà venire tracciata una ulteriore fiancata laterale destra o sinistra (con AREAFILL). Di conseguenza, i punti della penultima sequenza non saranno più necessari. Si dovranno perciò utilizzare i punti delle sequenze successive fino alla fine del loop z.

Se si è selezionato umrahm$ = "j", una parete anteriore apparterrà alle fiancate laterali. Il programma la traccia nello stesso modo come descritto. L'immagine è pronta, il programma attenderà la pressione del tasto, chiuderà lo schermo e la finestra e ritornerà al Workbench.

Proviamo ancora una volta altre funzioni con altri parametri. Un esempio: ottenere una panoramica della funzione tramite inserimenti di intervalli molto grandi e limitare in seguito i campi interessanti. Selezionare poi piccoli angoli di rotazione, piccole scale ed un osservatore molto distante (beo.z = -4000). Disinserire l'incorniciatura. Se abbiamo trovato le parti interessanti, proseguiremo per un raffinamento ulteriore (per motivi di velocità) impostando la precisione matematica a valori bassi.

5.3 Linee e superfici nascoste in spazi organizzati

Ciò che si nasconde dietro questo titolo non è così complicato come sembra. Negli ultimi Capitoli sono state mostrate le possibilità e le tecniche con le quali realizzare linee e superfici nascoste in maniera semplicissima e senza calcoli complicati in grafici di funzione tridimensionali.

Ora il problema riguarda le cosiddette: "Hidden Lines and Surfaces", generalmente uno degli aspetti più difficili nella grafica per computer. Esiste a questo scopo una serie molto ampia dei più diversi algoritmi che sono veloci o prendono in considerazione casi diversi di accostamento di oggetti e di superfici. E' molto difficile realizzarli entrambi contemporaneamente. Maggiore è il numero dei casi da prendere in considerazione, tanto più difficili e complicate saranno le routine per la soppressione delle superfici.

Poiché più avanti verremo a conoscenza di un procedimento che affronta tutte le eventualità al cento percento (il **Ray-Tracing**), ma è anche relativamente lento, cerchiamo di avere qui maggiore velocità. Ci confronteremo nella scelta di oggetti e superfici e otterremo una tecnica che raggiunge risultati utilizzabili in Real-Time (tempo reale).

Il nostro procedimento si divide in due passaggi:

— Soppressione del retro della superficie
— Selezione della profondità

Il primo passaggio si riferisce alle superfici all'interno di un oggetto. Il secondo si riferisce alle superfici, che possono trovarsi anch'esse dentro un oggetto e che inoltre possono esistere in oggetti diversi fra loro. Con ciò cercheremo di coprire il maggior numero di superfici possibili all'interno di ogni oggetto. Nel secondo passaggio si prenderanno in considerazione le superfici coperte all'interno di un oggetto non ancora esaminate e si determinerà la copertura di oggetti diversi l'uno sotto l'altro.

Tuttavia, in ogni caso, dovremo estendere il nostro modello di oggetti, linee e punti. Si dovrà rilevare ogni singola superficie di un oggetto. Una superficie viene definita attraverso le linee che la delimitano, che vengono determinate con i punti di inizio e di fine. Inoltre, un oggetto può venire determinato da diverse superfici.

a) Soppressione del retro di una superficie

Prendiamo in considerazione l'esempio molto semplice di un corpo, un parallelepipedo (ved. Fig. 5.6). Tale parallelepipedo è composto da sei facce, delle quali solo tre sono effettivamente visibili. La cosa importante da ricordare è che una faccia in questo caso è completamente visibile o non lo è per niente, poiché sono solo tali facce che vengono individuate dall'algoritmo. La visibilità di tali tre facce dipende dalla relativa prospettiva. Come sarà, perciò, possibile scoprire quali facce sono visibili e quali invece no?

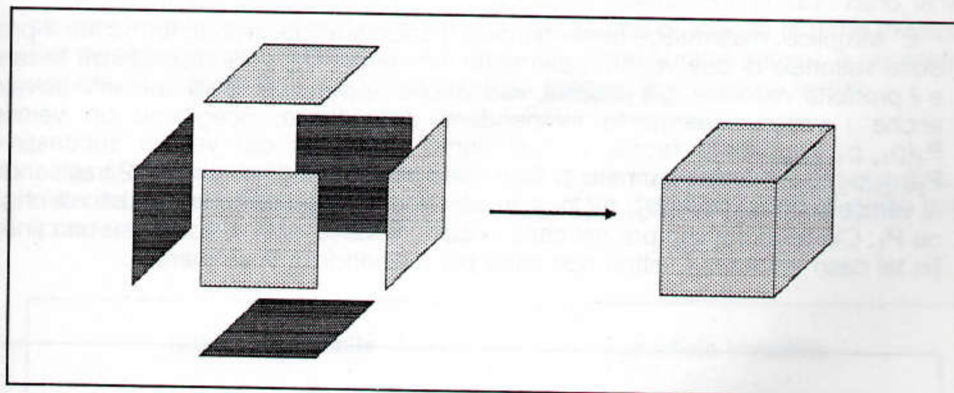


Fig. 5.6 Modello delle superfici di un cubo

A dir la verità tutto ciò non è poi così difficile. Abbiamo bisogno di ragionare sul concetto che i punti angolari (o gli spigoli) di una faccia sono numerati in senso orario. Osserviamo ogni singola faccia anteriormente e numeriamo i lati di conseguenza (ved. figura). Nel caso in cui un lato venisse ruotato rispetto all'osservatore, apparirebbe sullo schermo la faccia nella numerazione opposta. Proprio questo è il criterio secondo il quale una faccia è invisibile.

Ogni faccia possiede un lato visibile ed uno invisibile (quasi come uno specchio semiriflettente). Determineremo quale di questi due lati sarà rivolto verso di noi e quale no.

Ripercorriamo la via della matematica per cercare anche la direzione nella quale si mostrerà il lato visibile di una faccia. Mostriamo un vettore perpendicolare alla faccia. Punterà logicamente in tale direzione. (ved. Fig. 5.7).

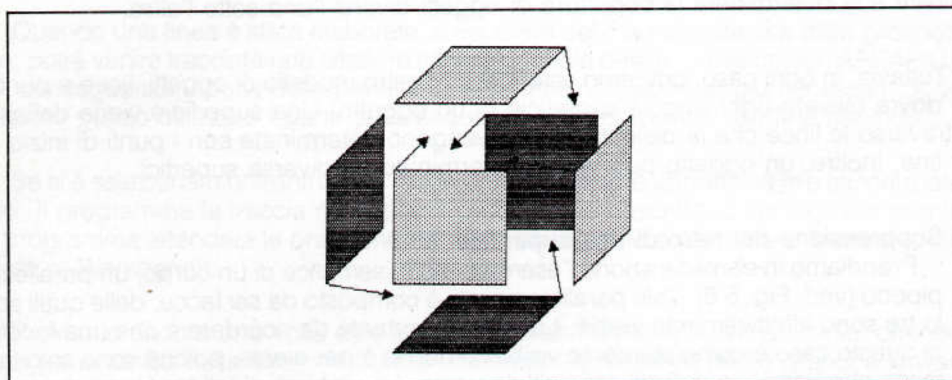


Fig. 5.7 Prodotto vettoriale di una superficie

E' semplice, matematicamente parlando, calcolare tale vettore, formando il prodotto vettoriale di due vettori linearmente indipendenti (per l'indipendenza lineare e il prodotto vettoriale, già descritti, ved. anche Appendice). Così abbiamo trovato anche i vettori linearmente indipendenti delle facce. Scegliamo un vertice $P_1(p_{1x},p_{1y},p_{1z})$ della faccia e costruiamo il vettore dal vertice successivo $P_2(p_{2x},p_{2y},p_{2z})$ - vettore numero 1. Otterremo invece il vettore numero 2 tracciando al vertice successivo $P_3(p_{3x},p_{3y},p_{3z})$ un secondo vettore dallo stesso punto di origine P_1. Ciò funziona sempre nel caso in cui i tre vertici non si trovino su una linea (in tal caso entrambi i vettori non sono più indipendenti linearmente).

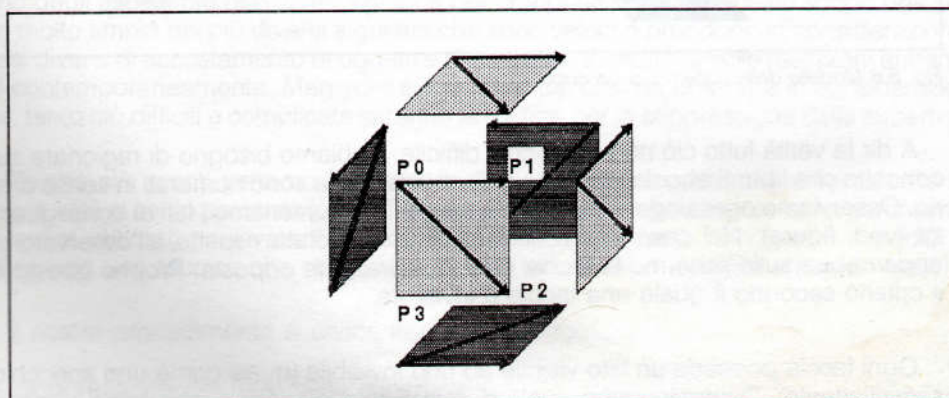


Fig. 5.8 Determinazione di vettori linearmente indipendenti di una superficie

Chiamiamo entrambi i vettori con $\vec{v}$ e $\vec{w}$:

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} p_{2x}-p_{1x} \\ p_{2y}-p_{1y} \\ p_{2z}-p_{1z} \end{pmatrix}$$

$$\vec{w} = \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} p_{3x}-p_{1x} \\ p_{3y}-p_{1y} \\ p_{3z}-p_{1z} \end{pmatrix}$$

Ora, per calcolare il vettore verticale $\vec{p}$, si dovrà determinare il prodotto vettoriale "$\vec{v}$ per $\vec{w}$" (da non confondere con il prodotto scalare):

$$\vec{p} = \vec{v} \times \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix}$$

$$= \begin{pmatrix} v_y{}^*w_z - v_z{}^*w_y \\ v_z{}^*w_x - v_x{}^*w_z \\ v_x{}^*w_y - v_y{}^*w_x \end{pmatrix}$$

Nel caso in cui tale vettore $\vec{p}$ di direzione (come osservatore) punti lontano, la faccia sarà visibile, mentre se punta verso di noi, stiamo gurdando la parte invisibile. Per determinare matematicamente in quale direzione punta il vettore, è necessario un vettore ulteriore e il cosiddetto prodotto scalare di entrambi i vettori.

Per quanto riguarda il secondo vettore, al quale vogliamo dare il nome di $\vec{s}$, si tratta del vettore dello sguardo, che dovrà puntare dall'osservatore al punto corrispondente da proiettare (ved. Fig. 5.9).

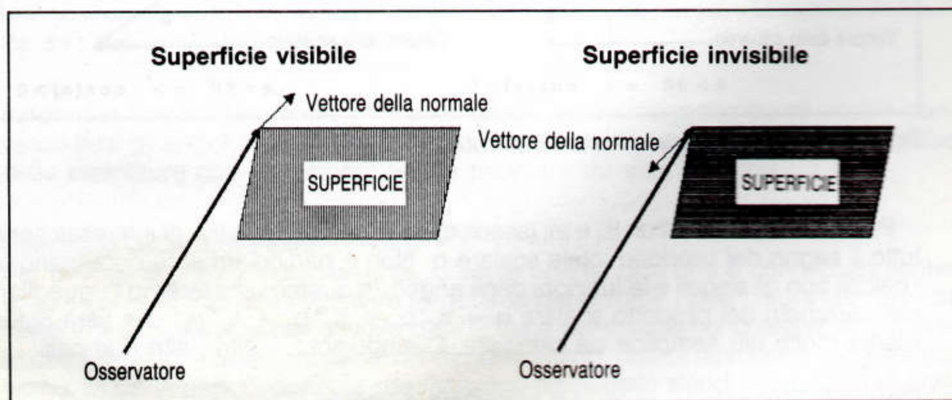


Fig. 5.9 Analisi della visibilità tramite vettore dello sguardo e vettore della direzione della superficie

Per $\vec{s}$ vale:

$$\vec{s} = \begin{pmatrix} s_x \\ s_y \\ s_z \end{pmatrix} = \begin{pmatrix} p_{1x}\text{-}oss_x \\ p_{1y}\text{-}oss_y \\ p_{1z}\text{-}oss_z \end{pmatrix}$$

con:

oss_x, oss_y, oss_z: Coordinate dell'osservatore
p_{1x}, p_{1y}, p_{1z}: Coordinate del punto di piede P_1 di $\vec{v}$ e $\vec{w}$

Se anche il vettore dello sguardo $\vec{s}$ e il vettore di direzione delle facce $\vec{p}$ puntano in direzioni uguali, la faccia sarà visibile, altrimenti no. Per ottenere un criterio matematico per la visibilità, l'algoritmo esamina l'angolo a tra i due vettori. Se per a vale: $-90 < a < 90$ entrambi i vettori puntano quasi nella stessa direzione e la faccia è visibile. Invece, se per a vale: $a >= 90$ oppure $a <= -90$, entra in gioco l'altro caso. Tramite il prodotto scalare q è possibile calcolare l'angolo tra i due vettori, che, come tutti sanno, fornisce un numero normale (e non un vettore come fa il prodotto vettoriale):

$$\begin{aligned} q &= \vec{s} * \vec{p} = |\vec{s}| * |\vec{p}| * \cos(a) \\ &= s_x * p_x + s_y * p_y + s_z * p_z \end{aligned}$$

Se a è nel campo da -90 gradi a +90 gradi, in questo caso il coseno cos(a) è positivo, altrimenti esso diventa uguale a zero (a = 90 oppure a = -90) oppure negativo (ved. Fig. 5.10).

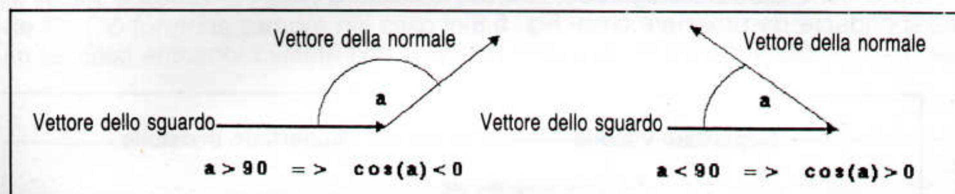


Fig. 5.10 Visibile o invisibile: questione dell'angolo

Poiché entrambi i fattori $|\vec{s}|$ e $|\vec{p}|$ (assoluti) sono sempre positivi, ci interessa soprattutto il segno del prodotto totale scalare q. Non è particolarmente necessario fare i calcoli con gli angoli e le funzioni degli angoli, in quanto utilizzeremo l'uguaglianza dei parametri del prodotto scalare $q = s_x * p_x + s_y * p_y + s_z * p_z$, che sarà naturalmente molto più semplice da calcolare. Distinguiamo, inoltre, altri due casi:

Caso nr. 1: $q > 0$ => La superficie è visibile
Caso nr. 2: $q \leq 0$ => La superficie è invisibile

L'algoritmo viene riassunto come segue:

1. Ogni oggetto ha una serie di superfici con lati numerati in senso orario
2. Ogni faccia viene sottoposta al controllo di visibilità
3. Determinazione di due vettori $\vec{v}$ e $\vec{w}$ linearmente indipendenti della superficie da esaminare tramite la scelta di tre punti P_1, P_2, P_3 che si susseguono l'uno dopo l'altro
4. Calcolo del vettore verticale $\vec{p}$ tramite il prodotto vettoriale $\vec{p} = \vec{v} \times \vec{w}$
5. Calcolo del vettore dello sguardo $\vec{s}$ risultante dall'osservatore -> P_1
6. Determinazione della direzione dei vettori $\vec{s}$ e $\vec{p}$ l'uno verso l'altro tramite determinazione del segno del prodotto scalare q di entrambi i vettori
7. Caso in cui $q > 0$: Superficie considerabile come visibile
   Caso in cui $q \leqq 0$: Superficie considerabile come invisibile

Questo algoritmo non funziona nel caso di superfici rivolte verso l'osservatore ma coperte da altre superfici dello stesso oggetto o da altri oggetti. Per impiegare da solo tale algoritmo, dovremo limitarci ad un oggetto (oppure disporre gli oggetti in

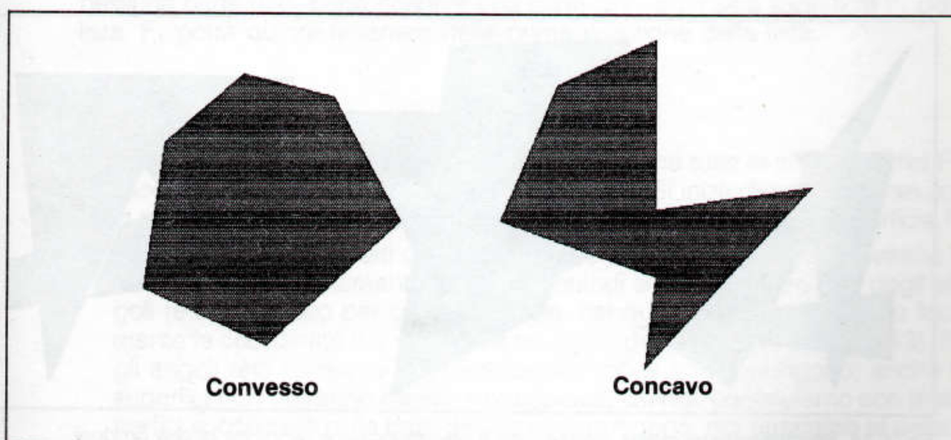


Fig. 5.11 Poligoni convessi e concavi

modo che non si intersechino in nessun caso). Inoltre, l'oggetto dovrà essere convesso (tutti gli angoli interni sono minori di 180 gradi). Un oggetto concavo avrebbe delle insenature che potrebbero venire ricoperte da altre superfici.

b) Selezione della profondità

Per ovviare a questo inconveniente, ci serve una tecnica che è conosciuta anche sotto il nome di Painters Algorithm (Algoritmo del pittore). Nella pittura tale tecnica viene spesso utilizzata per ovviare al problema delle superfici coperte, che esiste anche lì. Perciò, il pittore inizia con lo sfondo lontano, quindi dipinge un oggetto più vicino. In ciò le parti necessarie che facevano parte dello sfondo venivano ridipinte finché, finalmente, non si giungeva alle figure vicine, che ricoprivano nuovamente ciò che stava dietro.

Nella tecnica di computer le cose sono molto simili: prima si disegnano gli oggetti e le superfici posteriori, poi quelle anteriori che le potranno ricoprire completamente o quasi. La tecnica si adatta naturalmente solo al video, come possiede il nostro Amiga e la maggior parte dei computer. Per il plotter, per es., con il quale certamente non si può ridipingere una cosa sull'altra, vengono utilizzati gli algoritmi di Hidden-Line sopracitati, che dapprima eliminano le linee e le parti di linee coperte e quindi disegnano.

A dir la verità, tutto ciò non è una novità, poiché l'abbiamo già appreso con le funzioni tridimensionali. In quel caso era però molto più semplice trovare quale superficie stava dietro e quale stava davanti. Iniziavamo semplicemente a disegnare dalle coordinate z più elevate.

Con superfici normali nel modello dell'oggetto, invece, dovremo prima di tutto determinare quali superfici o parti di superficie sono davanti ad altre superfici. Potrà addirittura accadere che una parte di una superficie si trovi davanti e una parte dietro ad un'altra superficie, oppure che si intersechino due superfici (ved. Fig. 5.12).

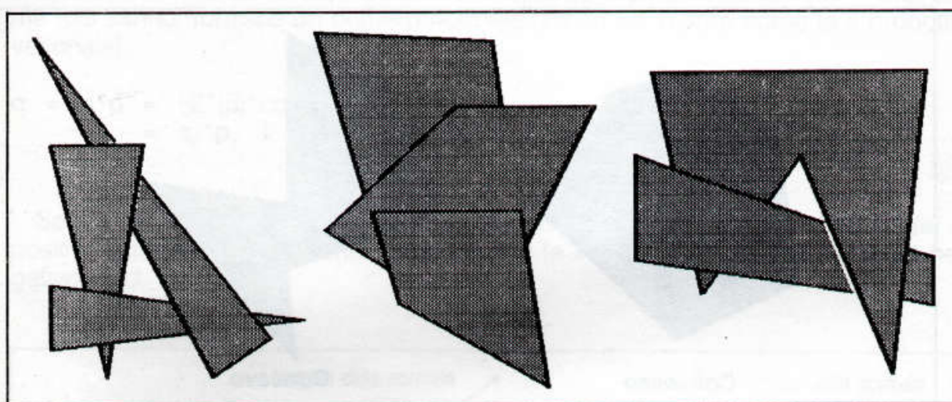


Fig. 5.12 Sovrapposizione ciclica, intersezione e sovrapposizione reciproca di due poligoni

Un algoritmo di selezione della profondità potrà essere complicato a piacere, a seconda dei casi che deve prendere in considerazione. Esso dovrà eventualmente suddividere le singole superfici per considerare chiaramente le priorità, ecc.

In questa tecnica si utilizzano solo quelle superfici che l'algoritmo sopra descritto ha lasciato come superfici visibili, in quanto quelle superfici determinate come invisibili sono invisibili in ogni caso. Con ciò diminuisce considerevolmente il numero delle superfici da elaborare (spesso di circa la metà).

Con l'algoritmo di selezione della profondità viene controllato se una o più superfici si sovrappongono. Il programmatore, perciò, cercherà di ottenere prima di tutto l'esclusione di più superfici possibili al fine di mettere in pratica metodi complicati per il minor numero di superfici possibile.

In generale si cerca di produrre una lista di priorità di superfici, che indicherà in quale sequenza esse dovranno venire tracciate. In questo caso esiste una tecnica chiara che effettua i seguenti passaggi. Il presupposto è che l'osservatore osservi lungo l'asse z. In caso diverso, questo scopo dovrà venire raggiunto tramite traslazioni e rotazioni corrispondenti (in maniera simile alla rotazione attorno ad un asse spaziale a piacere):

1. Di tutte le superfici verrà determinato il vertice dotato della coordinata maggiore z_{max} (il più lontana possibile) e quello che possiede la coordinata minore z_{min} (la più prossima all'osservatore).
2. Tutte le superfici verranno preselezionate secondo z_{max} decrescente. Nella prima posizione ci sarà la superficie con lo z_{max} più alto. Le chiameremo F_1, F_2,...F_n
3. Ora, la prima superficie F_1 dovrà venire confrontata con le altre superfici F_i (cioè: da F_2 a F_n):

   a) Nel caso in cui il punto più prossimo di F_1 (F_{1zmin}) venga a trovarsi più lontano del punto più lontano possibile della seconda superficie F_2 (F_{2zmax}), cioè:
   $F_{1zmin} \geqq F_{2zmax}$
   nessuna parte di F_1 potrà ricoprire una parte di F_2 o un'altra superficie F_i della lista. F_1 potrà quindi rimanere nella prima posizione della lista.

   b) Se invece vale:
   $F_{1zmin} < F_{2zmax}$
   sarà possibile che F_1 ricopra non solo F_2, ma anche tutte le altre superfici F_i. Per scoprire ciò, verrà effettuata una serie di test. Si inizierà naturalmente con i test più semplici e più veloci. Si dovrà testare F_1 con ogni altra superficie F_i:
   — Ricerca degli angoli retti compresi tra due superfici F_1 e F_i: si proiettano le due superfici normalmente sul piano, quindi si potrà trovare uno degli angoli retti compreso per ogni superficie, nel quale si determineranno facilmente le coordinate x e y minori e maggiori del vertice (ved. Fig. 5.13). Se gli angoli retti compresi di due superfici non si sovrappongono, anche le superfici non potranno essere sovrapposte. Quindi, continuiamo con le superfici successive nella lista. Se si sovrappongono, poi passiamo al prossimo test:

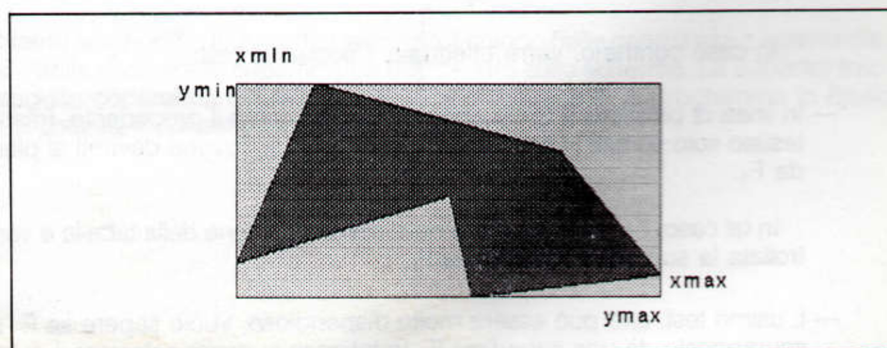


Fig. 5.13 Rettangolo contenente un poligono

— Tutti i vertici della superficie F_1 giacciono dietro ai piani tesi dalla seconda superficie F_i? Utilizziamo di nuovo il vettore $\vec{p}$ perpendicolare già calcolato nell'algoritmo sopracitato sul piano corrispondente F_i. Tale vettore punta, nel caso di tutte le superfici rimanenti, nella stessa direzione del vettore dell'osservatore (ved. sopra). Quindi punta lontano (ved. Fig. 5.14). Un vettore da un vertice a piacere della superficie F_i al vertice da esaminare di F_1 (chiamiamolo $\vec{f}$) punterà quindi nella stessa direzione di $\vec{p}$, qualora il vertice di F_1 si trovi dietro la superficie F_i. Se si trova davanti, esso punterà nella direzione opposta. Se $\vec{f}$ è invece perpendicolare a $\vec{p}$, il vertice di F_1 si troverà sul piano (non necessariamente sulla superficie!), che viene teso da F_i.

Determineremo il riferimento della direzione nella stessa maniera come per il test delle superfici con il prodotto scalare $\vec{f}*\vec{p}$, del quale ci interessa solo il segno.

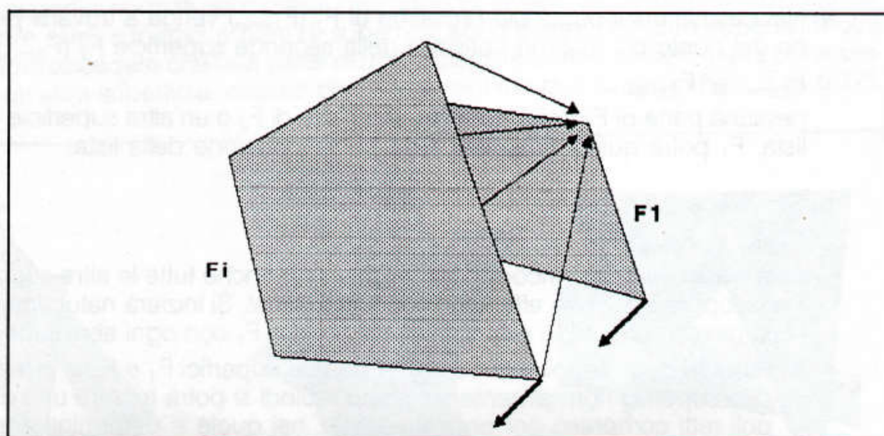


Fig. 5.14 Verifica della sovrapposizione di poligoni

Tutti i vertici di F_1 vengono testati in questa maniera. Se si trovano tutti dietro la superficie F_i in questione, si potrà continuare con la prossima superficie F_{i+1}.

In caso contrario, verrà effettuato il seguente test:

— In linea di principio si tratta dello stesso test come il precedente. Infatti, verrà testato solo se tutti i vertici della superficie F_i si trovano davanti al piano teso da F_1.

In tal caso, F_1 potrà rimanere nella prima posizione della tabella e verrà controllata la superficie successiva F_{i+1}.

— L'ultimo test, che può essere molto dispendioso, vuole sapere se F_1 si trova sovrapposto da una superficie F_i. In tal caso si dovrà calcolare il punto d'intersezione ecc.

Come abbiamo già detto, ogni superficie F_i viene sottoposta a questi test con riferimento a F_1. Se F_1 ha superato tutti i controlli, rimarrà nella prima posizione della lista (oppure verrà tracciata direttamente). La posizione di F_1 verrà quindi assunta in seguito dalla superficie F_2 successiva, ecc.

Se F_1 non supera un singolo test, esso verrà scambiato con quella superficie con la quale il test non ha funzionato. Con queste nuove superfici superiori verrà ripetuta l'intera procedura dall'inizio.

Contemporaneamente vengono memorizzate le vecchie posizioni di tale superficie. Infatti se si giungesse di nuovo allo scambio, ci troveremo in presenza di una sovrapposizione ciclica, che potrebbe venire risolta solo tramite suddivisione delle superfici.

Ancora più complicato, direte voi. Ma non scoraggiamoci, in quanto si tratta sempre di un algoritmo relativamente facile.

Se volessimo programmare tutti i passaggi di tale tecnica, dovremmo lasciar perdere la nostra intenzione di effettuare i movimenti in Real-Time (tempo reale). Lasciamo da parte per una volta la perfezione completa e accontentiamoci di un criterio di selezione della profondità semplice ed allettante che tuttavia prenda in considerazione una serie molto ampia di casi:

Determineremo facilmente un valore z intermedio di ogni superficie. Quindi, aggiungeremo il valore z a tutti i vertici delle superfici e divideremo il risultato per il numero dei vertici (un semplice calcolo di valore intermedio!). Questo risultato è naturalmente esatto solo per superfici che corrono parallele ai piani x-y. Tanto più una superficie penetra nello spazio, tanto maggiore sarà il pericolo che emergano degli errori, come vedremo in seguito: accoppiata con la ricerca per la visibilità, questa tecnica porta già a risultati sorprendenti (particolarmente in riferimento ai cosiddetti solidi di rotazione, che vedremo in seguito).

Se abbiamo selezionato le superfici secondo il criterio delle coordinate z intermedie, ci sarà più facile tracciare le superfici una dopo l'altra sullo schermo. Le superfici tracciate in seguito copriranno eventualmente quelle precedenti. Il programma in Basic che segue chiarirà il concetto di questa tecnica:

```
'***********************************
'**                               **
'**  Linee e superfici nascoste   **
'**                               **
'***********************************

PI = 3.141593

'Apertura nuovo schermo con finestra:
anz.farben% = 16
SCREEN 2,640,200,4,2
wx% = 631
wy% = 186
WINDOW 2,"Linee e  superfici  nascoste",(0,0)-(wx%,wy%),4+8,2

blauf = 0       'Fattore blu
gruenf = 0      'Fattore verde
rotf = 1        'Fattore rosso
blauadd = .1    'Aggiunta di blu
gruenadd = .1   'Aggiunta di verde
rotadd = 0      'Aggiunta di rosso

'Assegnazione colori:
FOR i=2 TO anz.farben%-1
  farbe = INT(i*100/15 + .5)/100
  rot   = farbe*rotf+rotadd
  gruen = farbe*gruenf+gruenadd
  blau  = farbe*blauf+blauadd
  IF rot>1 THEN rot=1
  IF gruen>1 THEN gruen=1
  IF blau>1 THEN blau=1

  PALETTE i,rot,gruen,blau
NEXT i

PALETTE 0,0,0,0     'Sfondo nero
PALETTE 1,1,.8,.13  'Colore cornice

' Parametri di trasformazione di Start
' Scala:
sx = 2
sy = 2
sz = 2

' Traslazione
tx = 5
ty = 5
tz = 5

' Rotazione
rx = -20
ry = 20
rz = 0
```

```
' Osservatore
bx = 0
by = 0
bz = 150

' Fonte di luce
lq.x& = -900
lq.y& = 1000
lq.z& = -500

' Intensita' luminosa/di superficie
li.int = .7
fl.int = 1
hin.int = .3

' Traslazione piano
tex = 300
tey = 90

' Scala piano
sex = 2
sey = 1

' Trasformazione in RAD dell'angolo di rotazione
rx = rx*PI/180
ry = ry*PI/180
rz = rz*PI/180

'Valore incremento rotazione (Gradi):
dig = 10

di = PI * dig/180                'Trasformazione in RAD

'Incremento osservatore:
binc = 20

POKE WINDOW(8)+27,1              'Colore dell'Area-Outline-Pen
flags = WINDOW(8)+32
POKEW flags,PEEKW(flags) OR 8 'Impostaz. dell'Area-Outline-Flag

'Lettura dati oggetto nelle matrici e
'definizione seguenti matrici:
'xr%(), yr%(), zr%() - coordinate del punto nello spazio
'xe(),ye(),ze()      - Coordinate trasformate
'fld%(,)             - Definizioni della superficie
'flz%(,)             - Superficie da tracciare
'anzeckd%()          - N.ro angoli di ogni superf. definita
'anzeckz%()          - Numero angoli di ogni superficie
'                      da tracciare
'sort.f%()           - Indici selezionati delle superf.
'mit.z()             - Valori z medi di matrice delle superf.
'farben()            - Intensita' colore di tutte le superf.
'                      da tracciare

get.objects
```

```
'Loop principale:
'****************

flag%=0
WHILE flag%<>1

  transform           'Trasformare tutti i punti
  verdecke            'Filtrare le superf. coperte
                      '(e determinarne i colori)
  projektion          'Proiettare tutti i punti

  CLS                 'Cancellazione finestra

  PATTERN &HFFFF      'Motivo di righe = a zig zag

  'Disegno oggeto:
  FOR i=0 TO afz%-1                         'Tutte le superfici
    flaech.nr% = sort.f%(i)                 'nr. superf. da tracciare
    FOR k=0 TO anzeckz%(flaech.nr%)-1       'Tutti i punti angol. superf.
      punkt.nr% = flz%(flaech.nr%,k)        'numero punto
      x% = tex + sex*xe(punkt.nr%)          'Coordinate del punto
      y% = tey - sey*ye(punkt.nr%)
      IF x%<0     THEN x% = 0
      IF x%>=wx%  THEN x% = wx%-1
      IF y%<0     THEN y% = 0
      IF y%>=wy%  THEN y% = wy%-1

      AREA (x%,y%)

    NEXT k

    COLOR INT(farbe(flaech.nr%)*14)+2 'Impostaz. valore colore
    AREAFILL 0                         'disegno superficie
  NEXT i

  maus% = 0 : flag%=0
  WHILE maus%<>1 AND flag%=0
    SLEEP          'Attesa evento

    'Assunzione delle coordinate del Mouse come
    'nuove coordinate di punto zero
    maus%=MOUSE(0)
    IF maus%=1 THEN
      tex = MOUSE(3)
      tey = MOUSE(4)
    END IF

    ch% = ASC(INKEY$+CHR$(0))
    IF ch%=31 THEN  'Cursore a sinistra =>
      ry=ry+di      'Aumento angolo rotazione asse y
      flag% = -1    'Flag per nuovo disegno
    END IF
    IF ch%=30 THEN  'Cursore a destra =>
```

```
  ry=ry-di         'diminuzione angolo rotazione asse y
  flag% = -1       'Flag per nuovo disegno
END IF
IF ch%=28 THEN   'Cursore in alto =>
  rx=rx+di         'Aumento angolo rotazione asse x
  flag% = -1       'Flag per nuovo disegno
END IF
IF ch%=29 THEN   'Cursore in basso =>
  rx=rx-di         'diminuzione angolo rotazione asse x
  flag% = -1       'Flag per nuovo disegno
END IF
IF ch%=127 THEN 'Del => Delimitazioni superfici si/no
  POKEW flags,PEEKW(flags) XOR 8
  flag% = -1
END IF
IF ch%=43 THEN   '+ => Scala verso l'alto
  sx = sx+1
  sy = sy+1
  sz = sz+1
  flag% = -1
END IF
IF ch%=45 THEN   '- => scala verso il basso
  sx = sx-1
  sy = sy-1
  sz = sz-1
  flag% = -1
END IF
IF ch%=56 THEN   '8 => Allontanamento osservatore
  bz = bz-binc
  flag% = -1
END IF
IF ch%=50 THEN   '2 => Avvicinamento osservatore
  bz = bz+binc
  flag% = -1
END IF
IF ch%=139 THEN 'Help => Rotazione = 0
  rx = 0
  ry = 0
  rz = 0
  flag% = -1
END IF
IF ch%=8 THEN    'Tasto indietro => cambio colore
  'Rotazione parametri colore:
  zwis     = rotf
  rotf     = gruenf
  gruenf   = blauf
  blauf    = zwis
  zwis     = rotadd
  rotadd   = gruenadd
  gruenadd = blauadd
  blauadd  = zwis
```

```
      'Assegnazione colori:
      'FOR i=2 TO anz.farben%-1
        farbe = INT(i*100/15 + .5)/100
        rot   = farbe*rotf+rotadd
        gruen = farbe*gruenf+gruenadd
        blau  = farbe*blauf+blauadd
        IF rot>1 THEN rot=1
        IF gruen>1 THEN gruen=1
        IF blau>1 THEN blau=1

        PALETTE i,rot,gruen,blau
      NEXT i
    END IF

    'Se finestra chiusa -> fine
    IF WINDOW(7)=0 THEN
      flag%=1
    END IF

  WEND
WEND

WINDOW OUTPUT 1
SCREEN CLOSE 2

END

'Lettura dati oggetto:

SUB get.objects STATIC

  'Lettura coordinate punto:
  SHARED ap%

  READ ap%            'Numero punti

  DIM SHARED xr%(ap%-1),yr%(ap%-1),zr%(ap%-1)
  DIM SHARED xe(ap%-1),ye(ap%-1),ze(ap%-1)

  FOR i=0 TO ap%-1
    READ xr%(i),yr%(i),zr%(i)
  NEXT i


  'Lettura definizione superfici:
  SHARED afd%, afz%

  READ afd%           'Numero superfici definite
  afz% = afd%         'Numero superfici da tracciare
  READ eckenmax%      'N.ro massimo di angoli e superfici
```

```
  DIM SHARED fld%(afd%-1,eckenmax%-1),flz%(afz%-1,eckenmax%-1)
  DIM SHARED anzeckd%(afd%-1),anzeckz%(afz%-1)
  DIM SHARED sort.f%(afz%-1),mit.z(afz%-1)
  DIM SHARED farbe(afz%-1)

  FOR i=0 TO afd%-1
    READ anzeckd%(i)   'Numero angoli di questa superficie
    FOR k=0 TO anzeckd%(i)-1
      READ fld%(i,k)
    NEXT k
  NEXT i

END SUB

'Trasformazione di tutti i punti spaziali:
SUB transform STATIC
  SHARED ap%
  SHARED sx,sy,sz,tx,ty,tz,rx,ry,rz

  'Calcolo costanti per la rotazione:
  si.x = SIN(rx) : co.x = COS(rx)
  si.y = SIN(ry) : co.y = COS(ry)
  si.z = SIN(rz) : co.z = COS(rz)

  A = co.y * co.z
  B = co.y * si.z
  C = -si.y
  D = si.x*si.y*co.z - co.x*si.z
  E = si.x*si.y*si.z + co.x*co.z
  F = si.x*co.y
  G = co.x*si.y*co.z + si.x*si.z
  H = co.x*si.y*si.z - si.x*co.z
  J = co.x*co.y

  FOR i=0 TO ap%-1
    ' Trasformazioni
    xt = sx*xr%(i) + tx          'Scala
    yt = sy*yr%(i) + ty          'e traslazione
    zt = sz*zr%(i) + tz

    xe(i) = xt*A + yt*B + zt*C   'Rotazioni
    ye(i) = xt*D + yt*E + zt*F
    ze(i) = xt*G + yt*H + zt*J
  NEXT i

END SUB

'Proiezione di tutte le coordinate spaziali sul piano:
SUB projektion STATIC
  SHARED ap%
  SHARED bx,by,bz
  FOR i=0 TO ap%-1
    ' Proiezione centrale:
    zwis = ze(i) - bz
    xe(i) = bx - bz * (xe(i)-bx)/zwis
    ye(i) = by - bz * (ye(i)-by)/zwis
  NEXT i
END SUB
```

```
'Filtraggio di tutte le superfici coperte
'e selezione dei valori z medi:

SUB verdecke STATIC
  SHARED afd%, afz%
  SHARED bx,by,bz

  afz% = 0            'Numero superfici da tracciare

  FOR i=0 TO afd%-1

    'Fase 1: Analisi dorso della superficie:
    '****************************************

    'Determinazione di due vettori indipendenti
    'linearmente della superficie:
    'v = P2-P1 // w = P3-P1:
    'dove: P1,P2,P3 = Punti angolari della superficie
    P1% = fld%(i,0)
    P2% = fld%(i,1)
    P3% = fld%(i,2)
    P1.x& = xe(P1%)
    P1.y& = ye(P1%)
    P1.z& = ze(P1%)
    v.x& = xe(P2%) - P1.x&
    v.y& = ye(P2%) - P1.y&
    v.z& = ze(P2%) - P1.z&
    w.x& = xe(P3%) - P1.x&
    w.y& = ye(P3%) - P1.y&
    w.z& = ze(P3%) - P1.z&
'Determinazione prodotto vettoriale p = v x w:
p.x& = v.y&*w.z& - v.z&*w.y&
p.y& = v.z&*w.x& - v.x&*w.z&
p.z& = v.x&*w.y& - v.y&*w.x&

'Calcolo del vettore dello sguardo s da osservatore a P1:
s.x& = P1.x& - bx
s.y& = P1.y& - by
s.z& = P1.z& - bz

'Calcolo prodotto scalare q = p * s :
q& = p.x&*s.x& + p.y&*s.y& + p.z&*s.z&

'Controllo segno di q:
IF q&>0 THEN

  'Identificazione della superficie come visibile:
  anzeckz%(afz%) = anzeckd%(i)

  sum.z = 0
  FOR k=0 TO anzeckd%(i)-1
    flz%(afz%,k) = fld%(i,k) 'Trasferimento definiz. superf.
    sum.z = ze(fld%(i,k)) + sum.z 'creazione della somma z
  NEXT k

  GOSUB farbe          'Ottenimento valore colore della superf.
```

```
'Fase 2: selezione valore medio z
'********************************

'Memorizzazione valore medio z in matrice z
mittel   = sum.z / (anzeckz%(afz%)-1)
mit.z(afz%) = mittel

'Ordinare l'indice superfici nella matrice di sort secondo z:
k = 0
'Ricerca punto selezione:
WHILE (mittel <= mit.z(sort.f%(k)) AND k < afz%)
  k=k+1
WEND
IF k <= afz% THEN
  'Spostamento a partire dal punto di selezione:
  FOR m=afz% TO k STEP -1
      sort.f%(m+1) = sort.f%(m)
    NEXT m
  END IF
  sort.f%(k) = afz%   'Annotazione indice della superf.

  afz% = afz%+1       'Incremento numero superf. visibili

END IF

NEXT i                    'Prossima superf.

EXIT SUB

'Ottenimento intensita' colore della superf.
farbe:
  SHARED lq.x&, lq.y&, lq.z&
  SHARED li.int, fl.int, hin.int

  ' Vettore dal punto della superf. alla fonte di luce
  L.x& = P1.x& - lq.x&
  L.y& = P1.y& - lq.y&
  L.z& = P1.z& - lq.z&

  ' Intensita' della luce incidente:
  cos.a = (p.x&*p.x& + p.y&*p.y& + p.z&*p.z&)
  cos.a = cos.a * (L.x&*L.x& + L.y&*L.y& + L.z&*L.z&)
  cos.a = (p.x&*L.x& + p.y&*L.y& + p.z&*L.z&)/SQR(cos.a)

  farb = hin.int*fl.int              'Intensita' di sfondo
  IF cos.a < 0 THEN                  'Superf. rivolta alla luce?
    farb = farb - li.int*fl.int * cos.a ' SI! determinare l'in-
  END IF                                ' cidenza della luce
  farbe(afz%) = farb - INT(farb) 'solo fra 0 e 1!
RETURN

END SUB
```

```
'Dati oggetto:
'************

'Coordinate tridimensionali del punto:

'Numero dei punti:
DATA 10

'Punti oggetto:
DATA  0, 0, 0,    6, 0, 0,    6,10, 0
DATA  0,10, 0,    3,15, 0,    3,15,15
DATA  6,10,15,    6, 0,15,    0, 0,15
DATA  0,10,15

'Definizioni superfici:

'Numero delle superfici:
DATA 9

'Numero max. di angoli di una superficie:
DATA 4

'Numero degli angoli + numero punti angolari delle superfici:
DATA 4,  3, 0, 1, 2
DATA 4,  2, 1, 7, 6
DATA 4,  6, 7, 8, 9
DATA 4,  9, 8, 0, 3
DATA 4,  0, 8, 7, 1
DATA 4,  4, 2, 6, 5
DATA 4,  5, 9, 3, 4
DATA 3,  3, 2, 4
DATA 3,  6, 9, 5
```

Nonostante la lunghezza del programma ci sorprenderemo nel vedere che abbiamo già appreso qualcosa. Conosciamo già molto dai Capitoli precedenti: le scale, le traslazioni, le rotazioni, le proiezioni centrali ecc., ecc., ecc.... Non abbiate paura di esaminare il programma, in ogni caso ne vale la pena!

Nel momento in cui il programma viene lanciato, apparirà, come al solito, un nuovo schermo sul video. Subito dopo vedremo la nostra vecchia casetta, questa volta un po' più perfezionata. Pensiamo ancora un po' al funzionamento prima di entrare completamente nel programma. Siamo di nuovo nella situazione di modificare la casetta (o ogni altro oggetto a piacere) tramite la pressione dei tasti. Abbiamo a disposizione i tasti seguenti:

<Tasto sinistro del Mouse>	Posizionamento dell'oggetto
<Cursore a sinistra>	Elevazione dell'angolo di rotazione attorno all'asse y
<Cursore a destra>	Diminuzione dell'angolo di rotazione attorno all'asse y
<Cursore su>	Elevazione dell'angolo di rotazione attorno all'asse x
<Cursore giù>	Diminuzione dell'angolo di rotazione attorno all'asse x

<Del>	Inserimento/disinserimento dell'incorniciatura della superficie
< + >	Ingrandimento dell'oggetto
<->	Riduzione dell'oggetto
<8>	Allontanamento dell'osservatore
<2>	Avvicinamento dell'osservatore
<Help>	Tutti gli angoli di rotazione = 0
<Spazio indietro>	Cambio di colori
<Gadget di chiusura>	Termine del programma

Non perdiamo altro tempo e passiamo a descrivere il programma:

All'inizio troviamo naturalmente tutte le solite inizializzazioni. Ci si è dedicati molto all'applicazione del colore. L'oggetto dovrà apparire in seguito, a seconda della pressione dei tasti, nei colori rosso, verde o blu (eventualmente con altre aggiunte di colori). Ogni colore è disponibile in 16 sfumature (da 0 a 15). Le 14 intensità più chiare vengono memorizzate in 14 registri di Palette, in modo da essere a disposizione in qualsiasi momento. I due registri inferiori sono riservati per i colori della cornice e per i colori di sfondo. Il programma ha istituito lo schermo relativo naturalmente con quattro gradazioni di colore (16 colori).

L'inizializzazione di colore descritta ha luogo nel primo loop FOR...NEXT del programma. Non irritiamoci ancora a causa di formule complicate. Qui verranno infatti convertiti i valori di intensità dei colori da 0 a 15 nei valori tra zero e uno (con arrotondamento), dei quali avrà bisogno l'AmigaBasic nel suo comando Palette. Inoltre, il programma calcolerà le intensità per ognuno dei tre colori di fondo. A questo punto si predisporranno le modifiche per renderlo più gradevole e piacevole, nel caso in cui i colori selezionati non fossero corrispondenti al proprio gusto.

A questo punto i valori di partenza vengono selezionati per la fase di messa in scala, di traslazione, di rotazione, per la posizione dell'osservatore, per la traslazione dei piani e la scala dei piani, nonché alcuni valori di incremento per il cambiamento conforme ai tasti di tali parametri. Anche in questo caso sarà il nostro campo di attività a trovarsi in "indagine" da parte di tale programma, in quanto si potrà regolare e governare nel vero senso della parola.

Prima di venire al nocciolo della questione, diamo ancora un chiarimento su di un passaggio interessante del programma:

```
POKE WINDOW(8)+27,1
flags = WINDOW(8)+32
POKEW flags,PEEKW(flags) or 8
```

Con il primo POKE regoliamo il colore del cosiddetto Pens OUTLINE sul registro di Palette 1 (per gli amici del C è forse già nota la funzione **SetOPen( )**). Questo "pastello" viene utilizzato da alcune funzioni grafiche del sistema operativo (per es. dai comandi di area), per tracciare per es. cornici di superfici. Per comunicare a tali funzioni

che dovranno tracciare immediatamente alcune cornici, si dovrà impostare il flag OUTLINE di area con il secondo POKE. In seguito nel programma scopriremo una seconda posizione, nella quale tale flag ritornerà ad essere cancellato come reazione della pressione di un tasto (in questo caso Del). Da ora in avanti, ogni superficie verrà tracciata senza cornice. Non potremo così far funzionare la funzione SetOPen( ), poiché questa rappresenta solo una macro in C e non è disponibile nella serie di comandi del sistema operativo.

Ed ora l'essenziale, finalmente! Introduciamo l'intera operazione tramite un salto alla routine get.objects. Come si apprende dal commento nel programma, a questo punto vengono installate numerose matrici e vengono letti dei dati dalle righe DATA, e così via. Non è difficile comprendere la routine stessa. Solo la funzione delle matrici dovrà venire analizzata accuratamente. I campi seguenti di inizializzazione hanno significato fondamentale per l'intero programma:

xr%( ),yr%( ),zr%( )	Coordinate spaziali di tutti i punti
xe( ),ye( ),ze( )	Matrici di calcolo per le coordinate del punto, in seguito: coordinate dei piani
fld%(,)	Tutte le superfici definite costituite dai numeri dei vertici
anzeck%( )	Numero degli angoli di tutte le superfici definite
flz%(,)	Successivamente: tutte le superfici da tracciare (!)
anzeckz%( )	Numero degli angoli di tutte le superfici da tracciare
sort.f%( )	Numerazione delle superfici selezionate
mit.z%( )	Profondità intermedie selezionate
farbe( )	Valori di intensità del colore di tutte le superfici

Tali matrici utilizzate parzialmente in seguito contengono tutte le informazioni che necessitano al programma, per tracciare tutti gli oggetti desiderati nel tipo e nel modo esatto. Impareremo in seguito la spiegazione di tutto ciò.

Come avremo forse già notato, in questo programma si è rinunciato ad alcune definizioni di linee. Nessuna meraviglia, avremo a che fare quasi esclusivamente con superfici. Un oggetto è costituito anche da un numero indefinito di superfici. Ognuna di esse, a sua volta, è costituita da diversi vertici (almeno tre), che vengono listati in una sequenza perfettamente chiara in senso orario (non dimentichiamo l'algoritmo delle superfici nascoste!). Ogni definizione di superficie è costituita da un numero, il quale indica il numero dei vertici (anzeck%( )) e da una matrice dei vertici. Per creare un oggetto dovremo, per prima cosa, concentrare tutti i vertici e predisporli nella matrice dei punti. In seguito numerare tutte le superfici di questo oggetto e listare superficie per superficie e vertice per vertice.

A questo punto, rivolgiamoci di nuovo al programma principale, che proseguirà con grandi loop principali. Si concluderà se chiuderemo la finestra di ingresso tramite il gadget CLOSE. Senza tanti preamboli, richiameremo le tre grandi routine che prepareranno tutti i dati spaziali e li predisporranno per il disegno:

transform	Trasformazione di tutti i punti spaziali
verdecke	Riconoscimento di tutte le superfici da tracciare e selezione della profondità (nonché calcolo del colore)
projektion	Conversione in coordinate piane del punto

A questo punto, non assisteremo più a nessuna emozione, in quanto abbiamo già conosciuto queste routine dagli altri programmi. Rotazione, traslazione e scala di tutti i punti spaziali appartengono ormai al passato.

Di estrema importanza nella routine è "scoperta"; questo sarà il nostro nuovo argomento! Per l'introduzione della ricerca del retro della superficie, il programma sceglierà due vettori del piano indipendenti linearmente. Tuttavia, ciò presuppone che si siano indicati correttamente i dati delle nostre superfici. Diventerà piuttosto complicato se dovremo fare in modo che tre vertici si trovino su una stessa linea. Di conseguenza, i vettori ricercati non saranno indipendenti linearmente. Risultato: confusione totale.

Vengono selezionati il vettore $\vec{v}$ dal primo vertice (P1) al secondo (P2) e il vettore $\vec{w}$ dal primo al terzo (P3). Un vettore si calcola naturalmente tramite la differenza di due vettori del punto. Nello stesso modo verrà determinato nella routine.

In seguito si dovrà determinare, in conformità all'algoritmo, il vettore perpendicolare alla superficie, tramite il prodotto vettoriale $\vec{p}=\vec{v}\times\vec{w}$, e poi il vettore dello sguardo $\vec{s}$ dall'osservatore al punto P1 e in seguito il prodotto scalare $\vec{q}=\vec{p}*\vec{s}$. A questo punto, potremo verificare il segno di q. Nel caso in cui questo sia negativo o uguale a zero, la superficie non sarà visibile (si potrà proseguire nella ricerca delle superficie successiva), e nel caso in cui questo sia invece positivo, potremo memorizzarla nella matrice che contiene tutte le superfici da tracciare: **flz%(,)**. Non si potrà naturalmente dimenticare la matrice del numero angolare **anzeckz%( )**. Di conseguenza, potremo volgere al termine la realizzazione dell'algoritmo per calcolare il retro delle superfici.

Successivamente incontreremo la chiamata di sottoprogramma "colori" con un piccolo intervallo che non appartiene alla struttura. Siccome dovremo impostare nell'AmigaBasic un piccolo valore per i tempi di esecuzione del programma, potremo chiudere un'occhio. Si tratta dell'attribuzione del colore. Naturalmente, sarà molto semplice indicare e preparare i colori per ogni superficie da tracciare. Il perfezionista dichiarerà che è necessario avere in qualche modo una sorgente luminosa per le diverse sfumature di colore e per aumentare l'effetto spaziale. In questo sottoprogramma calcoleremo con il nostro Amiga l'intensità della luce che incontreremo sulle superfici. Questo calcolo sarà diverso da superficie a superficie in quanto bisognerà tener conto degli angoli della luce (un problema simile a quello per l'osservatore). La formula utilizzata per ciò (una formula semplice e nessun algoritmo complicato) non dovrà in questo caso venire spiegata. Ciò accadrà infatti nel Capitolo del Ray-Tracing, il punto fondamentale del nostro libro, che non ci dovrà sfuggire. Consultiamolo di nuovo se ne vogliamo sapere di più. Per il calcolo della formula bisognerà conoscere il vettore perpendicolare alla superficie, che dovrebbe già essere stato calcolato nella ricerca della visibilità.

Il programma memorizza l'intensità di colore preparata (un valore tra zero e uno) nella matrice farbe( ) per l'utilizzo successivo.

A questo punto entra in azione la fase 2 del test di visibilità: le superfici vengono selezionate secondo la nostra priorità z. Come già detto, semplifichiamo la cosa determinando un valore z intermedio per ogni superficie. Esso viene calcolato dalla somma dei valori z di tutti i vertici divisa per il numero dei vertici. Il valore z determinato varrà naturalmente solo per il centro delle superfici.

Il ruolo della seconda fase sarà quello di selezionare le superfici in modo corrispondente al loro valore z intermedio. In seguito, tutte le superfici dovranno venire tracciate in sequenza con z in ordine discendente. La selezione ha luogo tramite semplice introduzione nella giusta posizione. Poi continueremo non con le definizioni di superficie, bensì solo con i numeri delle superfici. Nella matrice **sort.f%( )** è presente la numerazione di tutte le superfici nella sequenza nella quale esse dovranno venire tracciate.

Dopo ciò, avremo già realizzato tutto relativamente alla visibilità o invisibilità di una superficie, per cui dovremo ridedicarci al programma principale. La funzione successiva viene chiamata "proiezione". Non più difficoltà particolari, non più lunghezze particolari e non più incognite particolari: proiezione centrale. Risparmiamoci lunghe spiegazioni e arriviamo subito al nocciolo della questione.

Il passaggio successivo sarà finalmente il disegno: cancellare lo schermo, regolare il motivo delle linee, tracciare superficie per superficie sullo schermo.

Il numero delle superfici da tracciare si trova in **afz%**. La numerazione nelle giuste sequenze in **sort.f%( )**. Per le superfici attuali di volta in volta, il numero viene trasferito a flaech.nr%, che servirà come indice (oltre al numero di punto k) per la matrice **flz%(,)** della superficie. Da tale matrice si ricaverà solo numero del punto per numero del punto secondo **punkt.nr%**, che, da parte sua, viene utilizzato come indice per le matrici proiettate del punto xe( ) e ye( ).

Ogni punto verrà arrestato e trasmesso al comando **AREA**. Quando questa è completa, sarà stato preparato anche ogni vertice e il programma avrà impostato anche il colore esatto che sarà a disposizione nella matrice **farbe( )** per ogni superficie, e quindi tracciato: **AREAFILL**.

A questo punto tutte le superfici sono tracciate e l'immagine è pronta. Ora si attenderà l'indicazione dell'utente in modo che egli possa approntare, analizzare e introdurre le reazioni corrispondenti. Cosa vi posso ancora dire se non di iniziare il gioco dal principio, tutto in velocità Basic?

Tutto chiaro? Bene. precipitiamoci ora ad eseguire altri oggetti. Potrete ampliare il programma anche per le vostre singole necessità. Buon divertimento!

CAPITOLO 6

Più realisticamente:

perfetta grafica tridimensionale
con effetti di luce,
riflessione e ombreggiature
tramite Ray-Tracing,
completo utilizzo dei colori dell'Amiga.

Ciò che il nostro elaboratore ha compiuto fin'ora per quanto riguarda la grafica è certamente notevole. Altrettanto notevole è naturalmente anche la velocità dell'elaboratore, che esegue i calcoli più complessi e ciononostante riesce, in tempo reale, a ruotare sullo schermo tutti gli oggetti, a ingrandirli, spostarli etc. Ciò che fin'ora ci ha colpito di più è sicuramente la verosimiglianza con tutti i fattori che costituiscono la realtà. Nella relatà infatti ci sono anche luci e ombre, gli oggetti riflettono la luce, sono opachi o lucidi, trasparenti o a specchio, hanno i colori più svariati dal blu al bianco, dal rosso al verde. Il fatto che tali effetti siano realizzabili solo con il massimo sforzo, applicando complessi metodi matematici, con i modelli visti fin'ora, non ha bisogno di venire ricordato; basta pensare all'algoritmo delle superfici nascoste. Ora vogliamo imparare invece una tecnica che risolva questi aspetti in maniera molto più semplice, anche se più lenta. Con questa tecnica anche gli effetti più complicati, quali il problema delle linee nascoste o delle ombre e luci, diventano un gioco da ragazzi. Tale tecnica ha un nome: Ray-Tracing.

6.1. Principi matematici del Ray-Tracing

Osservando il Ray-Tracing si nota immediatamente che si fa un uso molto intenso del calcolo vettoriale che abbiamo già incontrato in precedenza. Diamo quindi un'altra occhiata ai capitoli che ne parlano (Cap.: "Basi matematiche per i calcoli tridimensionali" oppure Appendice). Senza il calcolo vettoriale infatti non riusciremmo a venire fuori.

La parola Ray-Tracing significa qualcosa come "Inseguimento del raggio". Infatti in questo caso i raggi di luce vengono inseguiti nel loro percorso dalla fonte di luce, attraverso i veri oggetti, fino all'occhio dell'osservatore. Tutto ciò con una piccola differenza: l'inseguimento va nella direzione opposta. Esso infatti inizia al punto d'arrivo del raggio luminoso, cioè dall'osservatore, e ripercorre all'indietro il percorso del raggio, attraverso gli oggetti fino alla fonte luminosa. Con ciò siamo sicuri che verrranno tenuti in considerazione solo quei raggi di luce che colpiscono veramente l'occhio (o la macchina fotografica ecc.).

Tecnicamente, ciò accade così: ipotizziamo di stare seduti davanti a uno schermo e di osservare il mondo retrostante attraverso esso (stiamo ipotizzando che lo schermo sia composto solo da una lastra di vetro). Tale lastra di vetro può quindi venire vista come piano di proiezione, (Ved. Fig. 6.1), sulla quale è rappresentato il mondo retrostante.

Nel nostro procedimento, dovremo attribuire ad ogni punto di questo schermo il colore adeguato, per un totale, per esempio, di 320 × 256 punti, se operiamo in modo bassa risoluzione. Il calcolo del colore di ogni singolo punto è compito nostro. Immaginiamo un raggio dell'occhio dell'osservatore che passa attraverso il punto da calcolarsi dello schermo e va nel mondo retrostante. Esso si perderà nell'infinità dello spazio oppure incontrerà uno o più oggetti che si trovano nel mondo.

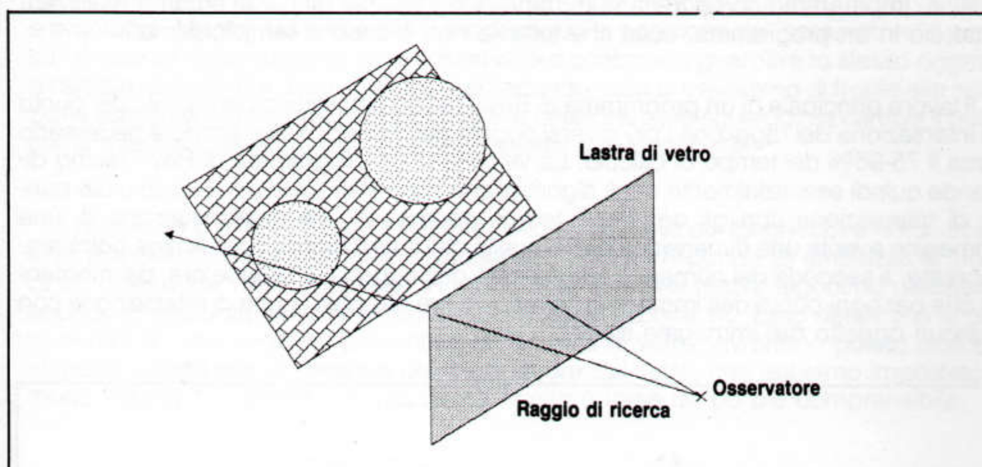


Fig. 6.1 Principio del RayTracing

Nel primo caso, il punto assumerà semplicemente il colore dello sfondo. Nel secondo caso invece, assumerà il colore dell'oggetto incontrerà per primo, cioè quello più vicino all'osservatore (e con ciò abbiamo quasi risolto il problema delle linee e superfici nascoste). Ciò che accade in seguito dipende dalla conformazione dell'oggetto. Al fine di rendere più interesssante questa tecnica, vediamo subito un paio di esempi:

Se nello spazio non esiste una fonte di luce ben determinata ma c'è solo un chiarore diffuso e l'oggetto è una semplice immagine blu opaca, il colore di quel punto specifico sulla lastra di vetro sarà il blu. L'oggetto, però, può anche essere a specchio. In questo caso dovrà venire determinato un secondo raggio, che dovrà venire calcolato secondo la legge "angolo di incidenza uguale all'angolo di uscita". Questo raggio potrà di nuovo disperdersi nello spazio o incontrare un altro oggetto. Se incontra un oggetto normale, il colore di tale oggetto verrà assunto come colore del punto, ma, se anche questo secondo oggetto è a specchio, il processo continuerà verso un altro e così via.

L'oggetto, però, può anche essere trasparente. Quindi il raggio verrà rifratto dalla superficie secondo la legge della rifrazione, ne uscirà dall'altra parte e continuerà verso l'oggetto successivo.

Forse però c'è anche una o più fonti di luce. In questo caso verrà tracciato un ulteriore raggio dal punto nel quale il raggio originale ha incontrato l'oggetto, fino alla fonte di luce. In ciò, l'angolo di questo raggio rispetto alla superficie dell'oggetto dipenderà dalla presenza o meno di un altro oggetto fra lui e la fonte di luce (ombreggiatura).

Stiamo scoprendo come ottenere effetti complicatissimi con mezzi semplici che approfondiremo nelle pagine seguenti, descrivendoli anche da un punto di vista mate-

matico. Impareremo come trattare le forme di oggetti più disparati e come realizzare tutto ciò in un programma, cosa che tuttavia non è ancora semplicissima.

Il lavoro principale di un programma di Ray-Tracing è costituito dal calcolo del punto di intersezione dei raggi con i più diversi oggetti del mondo. A tale scopo è necessario circa il 75-95% del tempo di calcolo. La velocità di un programma di Ray-Tracing dipende quindi essenzialmente dagli algoritmi che sono stati scelti per il calcolo del punto di intersezione con gli oggetti. Il tempo necessario alla determinazione di una immagine avente una dimensione pari a quella dell'intero monitor dell'Amiga potrà ammontare, a seconda del numero e complessità dell'oggetto, a diverse ore, dal momento che per ogni punto dell'immagine devono venire calcolati i punti di intersezione con ciascun oggetto dell'immagine stessa.

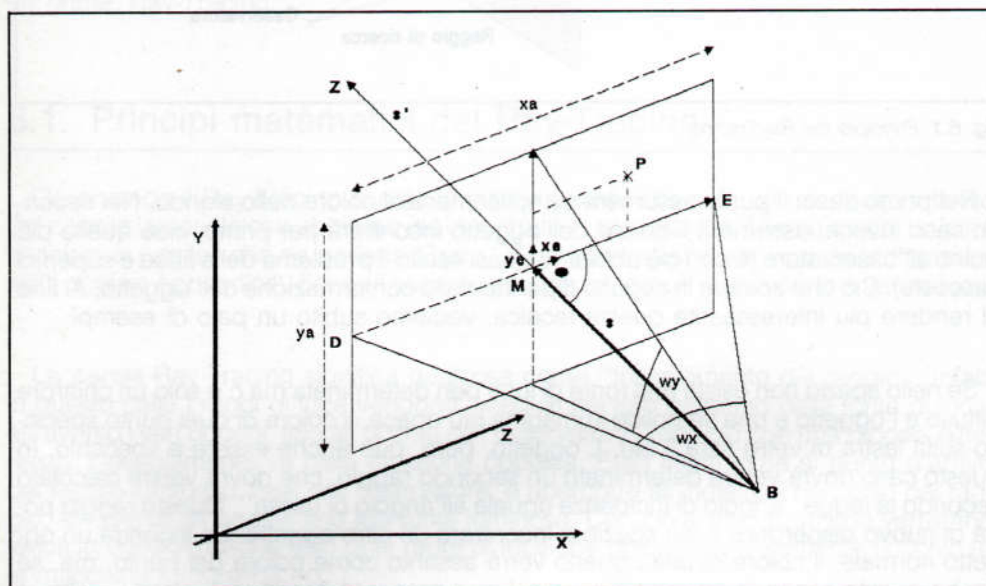


Fig. 6.2 Rappresentazione matematica del RayTracing

Per la parte matematica, osserviamo la Fig. 6.2. In essa le basi del Ray-Tracing sono ben evidenziati. Vediamo l'osservatore al punto B (b_x,b_y,b_z). La sua posizione, nel sistema di coordinate spaziali, è nota. Inoltre è noto il punto dello spazio al quale l'osservatore è volto: Z (z_x,z_y,z_z). Può trattarsi naturalmente di qualunque punto a piacere nel paesaggio, a seconda di quale oggetto inseriremo in seguito al centro della lastra di vetro. In ogni caso, esso non può coincidere con l'osservatore B.

Il vettore da B a Z, chiamato $\vec{s}'$, è quindi il cosiddetto vettore (allungato) dello sguardo,cioè il vettore che indica la direzione nella quale l'osservatore sta guardando. Esso sarà, in ogni caso, perpendicolare alla lastra di vetro. Inoltre deve essere nota la distanza dell'osservatore della lastra di vetro. Si tratta della distanza tra B ed M (M è il punto centrale del vetro). In alternativa potremo indicare direttamente il vettore allun-

gato dello sguardo o il punto cui si sta guardando. Ambedue hanno i loro vantaggi e svantaggi. Mentre nel primo caso, modificando la posizione dell'osservatore, il punto cui si guarda resta costante, cioè l'osservatore continua a guardare lo stesso oggetto, e cambia la direzione dello sguardo, nel secondo caso ci troveremo di fronte alla situazione esattamente contraria: resta invariata la direzione dello sguardo ma cambia il punto cui si guarda. Lasciamo quindi aperte queste due possibilità.

Il vettore da B a M si chiama $\vec{s}$ (vettore dello sguardo) la cui lunghezza è notoriamente $|\vec{s}|$ ($\vec{s}$ assoluto) e non può essere uguale a z. Esso corrisponde all'ampiezza di fuoco di un obiettivo e viene impostato in modo fisso a 1. Nel nostro algoritmo utilizziamo ancora, in aggiunta, un cosiddetto angolo di osservazione W_x. Esso determina la parte di mondo che verrà rappresentata sulla lastra di vetro. Avremmo potuto ottenere lo stesso effetto con la distanza dell'osservatore dal vetro, che abbiamo impostato in modo fisso a 1. Tuttavia il modo da noi scelto è forse un pò più comprensibile.

Il programma dovrà inoltre conoscere un paio di altri parametri: la risoluzione x e d y, cioè xa ed ya, nonché le grandezze x ed y del punto, cioè xpg ed ypg, al fine di non deformare l'immagine. Da tali parametri viene calcolato anche l'angolo di osservazione wy, che tuttavia non è necessario. xa ed ya indicano propriamente solo il numero dei punti da rappresentare nelle direzioni x ed y, cioè praticamente la dimensione della finestra. Più i valori per essi scelti sono piccoli, minore sarà il numero di punti da calcolare e maggiore sarà la velocità di elaborazine dell'immagine. Inoltre, per semplificare il calcolo, stabiliamo che la lastra di vetro non venga ruotata attorno all'asse z. I punti angolari di destra e di sinistra avranno quindi la stessa coordinata y l'uno rispetto all'altro, mentre quelli superiori e inferiori avranno, l'uno rispetto all'altro, delle coordinate x uguali. Questa decisione non è affatto tragica, in quanto nessuno vorrà inclinare il proprio monitor.

Calcoliamo dapprima il vettore $\vec{s}$. Conosciamo solo la sua lunghezza $|\vec{s}|$, mentre la sua direzione è la stessa di $\vec{s}'$ (vettore allungato dello sguardo. Quindi vale:

$$\vec{s} = \vec{n} * \vec{s}'$$

e, secondo la difinizione delle lunghezze dei vettori:

$$|\vec{s}|^2 = s_x^2 + s_y^2 + s_z^2$$

Il vettore s' è facilmente determinabile dai punti terminali B e Z (notare che i punti possono anche venire visti come vettori dal punto zero al punto in questione):

$$B + \vec{s}' = Z \qquad <=> \qquad \vec{s}' = Z\text{-}B$$

Quindi vale anche

$$\vec{s} = n * (Z\text{-}B)$$

e perciò

$|\vec{s}| = n * |Z\text{-}B| \quad <=>$

$$n = \frac{|\vec{s}|}{|Z\text{-}B|}$$

Quindi otterremo

$$\vec{s} = \frac{|\vec{s}|}{|Z\text{-}B|} * (Z\text{-}B)$$

La lunghezza $|Z\text{-}B|$ del vettore (Z-B) è facilmente determinabile tramite la nota formula della lunghezza:

$$|\vec{v}|^2 = v_x^2 + v_y^2 + v_z^2$$

$|\vec{v}|$ viene quindi determinato dalla radice della parte destra dell'equazione. Dal momento che $|\vec{s}|$ è sempre uguale a 1, l'equazione è notevolmente semplificata. Con ciò determiniamo il vettore dello sguardo $\vec{s}$ vero e proprio.

Al fine di indicare un punto P a piacere sulla lastra di vetro, utilizzeremo (naturalmente oltre al punto di osservazione e al vettore dello sguardo) i due vettori unitari $\vec{xe}$ ed $\vec{ye}$. Essi forniscono rispettivamente la distanza e ed y di due punti. La distanza dipende naturalmente anche dalle grandezze del punto xpg ed ypg. Un punto P con le coordinate x,y viene quindi determinato come segue:

$$P = B + \vec{s} + (x*xpg) * \vec{xe} + (y*ypg) * \vec{ye}$$

Tenere presente che il punto zero del sistema di coordinate dello schermo si trova nel suo centro M. Quindi x ed y hanno valori da -xa/2 fino a +xa/2 oppure -ya/2 fino a +ya/2 (xa ed ya sono rispettivamente le risoluzioni e ed y dello schermo).

Il calcolo dei vettori unitari $\vec{xe}$ ed $\vec{ye}$ ci interessa in modo particolare, anche se è un po' più complicato. Determiniamo dapprima $\vec{xe}$: dal momento che il vettore dello sguardo $\vec{s}$ deve essere perpendicolare alla lastra di vetro, il prodotto scalare $\vec{s}*\vec{xe}$ deve essere uguale a zero. Con ciò, per la direzione di $\vec{xe}$, abbiamo una definizione:

$\vec{s}*\vec{xe} = 0 \quad <=>$

$s_x*xe_x + s_y*xe_y + sz*xe_z = 0$

Come secondo criterio relativo alla lunghezza di $\vec{xe}$, vale:

$ME = (xa/2) * xpg * |\vec{xe}|$

dove

ME Distanza fra il centro e il punto E
xa Risoluzione x
xpg Grandezza punto x
$|\vec{xe}|$ Lunghezza del vettore unitario x.

Ambedue i criteri devono essere adempiuti. Cominciamo con la lunghezza del vettore unitario $|\vec{xe}|$:

Osserviamo ancora una volta il disegno. Il triangolo formato dai punti B, M ed E è rettangolo. L'angolo retto si trova nel punto M. L'angolo a B è mezzo angolo di osservazione $w_x/2$. Se cerchiamo la lunghezza della distanza ME, vale:

$$\tan(w_x/2) = ME / |\vec{s}| \quad <=>$$
$$ME = \tan(w_x/2)*|\vec{s}|$$

Per la lunghezza della distanza ME abbiamo già una definizione:

$$ME = (xa/2)*xpg*|\vec{xe}|$$

e uguagliando le due equazioni, otterremo:

$$(xa/2)*xpg*|\vec{xe}| = \tan(w_x/2)*|\vec{s}| \quad <=>$$

$$|\vec{xe}| = \frac{2*\tan(w_x/2)*|\vec{s}|}{xa*xpg}$$

E con ciò abbiamo determinato la lunghezza del vettore unitario x.

Dedichiamoci ora alla determinazione della direzione di $\vec{xe}$. Il punto di partenza è, anche in questo caso, l'equazione di cui sopra:

$$s_x*xe_x + s_y*xe_y + s_z*xe_z = 0$$

Ricorderemo senz'altro di aver stabilito l'angolo di rotazione della lastra di vetro attorno all'asse z come uguale a zero. Di conseguenza la coordinata y del piede di $\vec{xe}$ è identica a quella della punta (il vettore è parallelo al piano xz). Quindi varrà anche:

$$xe_y = 0$$

L'equazione viene semplificata con

$$s_x*xe_x + s_z*xe_z = 0$$

Qui però dobbiamo purtroppo distinguere tre casi:

Caso 1: $s_x <> 0$:
In questo caso possiamo risolvere semplicemente l'equazione secondo xe_x:

$$xe_x = -(s_z * xe_z)/s_x$$

Caso 2: $s_x = 0$ e $s_z <> 0$:
Dal momento che la divisione per zero è vietata, non è possibile risolvere l'equazione secondo xe_x. D'altra parte non possiamo introdurre un valore a piacere per xe_x, dal momento che deve venire adempiuto anche il secondo criterio (lunghezza del vettore $\vec{xe}$). Aiutiamoci quindi con la soluzione secondo xe_z:

$$xe_z = -(s_x * xe_x)/s_z$$

Caso 3: $s_x = 0$ e $s_z = 0$:
Qui abbiamo veramente le mani legate. Non ci è permessa nessuna delle due soluzioni presentate, ma ciò non è poi tragico, in quanto il vettore $\vec{s}$ in questo caso corre parallelo all'asse z (e siccome non può essere il vettore zero, in questo caso s_y è diverso da zero!). Il vettore unitario $\vec{xe}$ che stavamo cercando dovrà quindi correre parallelo all'asse z. xe_z è quindi uguale a zero e lo stesso vale per xe_y), mentre xe_x sarà uguale alla lunghezza del vettore unitario:

$$xe_z = 0$$
$$xe_z = |\vec{xe}|$$

Consideriamo ed approfondiamo invece i casi 1 e 2. s_x ed s_z sono noti (vettore dello sguardo). Dobbiamo quindi determinare l'altra incognita, cioè xe_z per il caso 1 ed xe_x per il caso 2. In ciò ci aiuterà la lunghezza del vettoe $\vec{xe}$, che siamo già in grado di calcolare:

$$|\vec{xe}|^2 = xe_x^2 + xe_y^2 + xe_z^2$$

e, con $xe_y = 0$ (vedi sopra):

$$|\vec{xe}|^2 = xe_x^2 + xe_z^2$$

Ora può venire applicata l'equazione vista in precedenza. Facciamolo per il primo caso (anche s_x è diverso da zero):

$$|\vec{xe}|^2 = xe_z^2 + (-s_z * xe_z/s_x)^2 \qquad <=>$$
$$|\vec{xe}|^2 = xe_z^2 + xe_z^2 * (s_z/s_x)^2 \qquad <=>$$
$$|\vec{xe}|^2 = xe_z^2 * [1 + (s_z/s_x)^2] \qquad <=>$$

$$xe_z^2 = \frac{|\vec{xe}|^2}{1 + (s_z/s_x)^2} \quad \text{(per caso 1: } s_x <> 0\text{)}$$

E con ciò abbiamo un'espressione per xe_x che potremo utilizzare nell'espressione incontrata in precedenza per xe_x.

Per il secondo caso otterremo analogamente:

$$xe_x^2 = \frac{|\vec{xe}|^2}{1 + (s_z/s_x)^2} \quad \text{(per caso 2: } s_x = 0 \text{ e } s_z <> 0\text{)}$$

Nello stesso modo determiniamo anche il vettore unitario y. Anche qui si devono distinguere tre casi e anche per questo vettore la coordinata x è uguale a zero. Arriviamo alle seguenti equazioni:

Caso 1: $s_y <> 0$:

$$\vec{ye}_y = -s_z * ye_z/s_y$$

$$ye_z^2 = \frac{|\vec{ye}|^2}{1 + (-s_z/s_y)^2}$$

Caso 2: $s_y = 0$ e $s_z <> 0$:

$$ye_z = -s_y * ye_y/s_z$$

$$ye_y^2 = \frac{|\vec{ye}|^2}{1 + (-s_y/s_z)^2}$$

Caso 3: $s_y = 0$ e $s_z = 0$:

$$ye_z = 0$$
$$ye_y = |\vec{ye}|$$

Per tutti i casi vale:

$$|\vec{ye}| = |\vec{xe}|$$

Naturalmente la lunghezza dei due vettori unitari è uguale, per cui la si dovrà calcolare una volta sola all'inizio.

Fig. 6.3 Calcolo dei raggi

Con ciò abbiamo posto le basi vere e proprie. Nel programma verrranno quindi inseriti tutti i punti da -xa/2 a +xa/2 e da -ya/2 a +ya/2, in due grossi Loop di FOR...NEXT nidificati. Con la formula già precedentemente illustrata:

$$P_m = B + \vec{s} + x{*}xpg{*}\overrightarrow{xe} + y{*}ypg{*}ye$$

si calcola il singolo punto della lastra di vetro immaginaria (ved. Fig. 6.3). Attraverso tale punto deve venire tracciato un raggio dall'osservatore B verso lo spazio. Tale raggio (una retta) verrà calcolato secondo la forma vettoriale dell'equazione di una retta:

$$P = P_m + k{*}\vec{b}$$

dove

$$\vec{b} = P_m - B$$

per cui:

$$P = P_m + k{*}(P_m - B)$$

dove:

P_m punto sulla lastra di vetro
B osservatore
$\vec{b}$ vettore di base da B a P_m
K fattore di allungamento
P un punto sulla retta

A questo punto per tutti gli oggetti situati nello spazio si dovrà controllare se vengono intersecati o toccati dalla retta. Se ciò accade per più oggetti, per gli ulteriori calcoli verrà utilizzato il punto di intersezione più vicino all'osservatore, cioè quello che possiede il K più piccolo. Come determinare i singoli punti di intersezione per i diversi oggetti è materia del prossimo paragrafo.

6.2. Calcolo del punto di intersezione con corpi e superfici.

6.2.1. Sfera

Il più semplice e più veloce calcolo del punto di intersezione è quello fra la retta dello sguardo ed una sfera. Di conseguenza la sfera è l'oggetto maggiormente utilizzato per il Ray-Tracing.

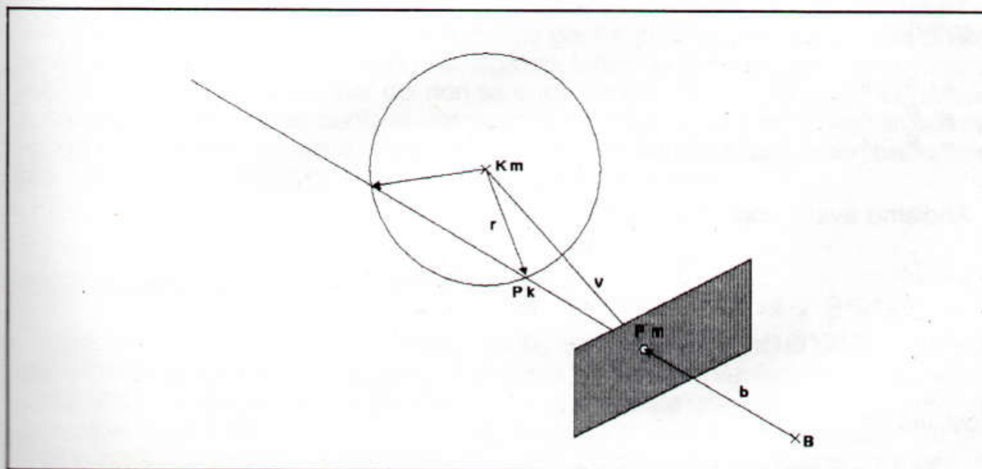


Fig. 6.4 Calcolo del punto di intersezione con una sfera

Osserviamo la Fig. 6.4. Stiamo cercando il vettore del raggio $\vec{r}$, la cui lunghezza corrisponde al raggio della sfera che ha come centro il punto K_m. d avrà il piede nel centro della sfera K_m e la punta sulla retta nel punto P_k:

$\vec{r} = P_k - K_m$

P_k è un punto della retta che attraversa il centro della lastra P_m con il vettore di direzione $\vec{b}$:

$$P_k = P_m + k*\vec{b}$$

quindi

$$\vec{r} = P_m + k*b - K_m \quad <=>$$
$$\vec{r} = (P_m - K_m) + k*b$$

Viene quindi cercato il fattore di allungamento K che fornisce il vettore del raggio $\vec{r}$. Poiché la lunghezza di r è il raggio della sfera, vale:

$$|\vec{r}|^2 = R^2$$

Utilizziamo la formula di cui sopra per r:

$$|(P_m-K_m)+k*\vec{b}|^2 = R^2$$

Per il vettore (P_m-K_m) utilizziamo per semplicità $\vec{v}$. Quindi varrà:

$$v = P_m-K_m$$

e quindi:

$$v_x = P_{mx}-K_{mx}$$
$$v_y = P_{my}-K_{my}$$
$$v_z = P_{mz}-K_{mz}$$

Andiamo avanti con:

$$|\vec{v}+k*\vec{b}|^2 = R^2 \quad <=>$$
$$\vec{v}^2 + 2*k*\vec{v}*\vec{b} + k^2*\vec{b}^2 = R^2 \quad <=>$$
$$(\vec{b}^2)*k^2 + (2*\vec{v}*\vec{b})*k + (\vec{v}^2-R^2) = 0$$

Sostituiamo:

$$a = \vec{b}^2 = b_x^2 + b_y^2 + b_z^2$$
$$b = 2*\vec{v}*\vec{b} = 2*(v_xb_x + v_yb_y + v_zb_z)$$
$$c = \vec{v}^2-R^2 = v_x^2 + v_y^2 + v_z^2 - R^2$$

ed otterremo:

$$a*k^2 + b*k + c = 0$$

Questa equazione di II grado potrà venire risolta secondo K. Utilizziamo la formula di soluzione delle equazioni di II grado:

$$k_{1,2} = \frac{-b +/- \sqrt{b^2 - 4*a*c}}{2*a}$$

Teniamo quindi presente che dovremo utilizzare le espressioni di cui sopra per i coefficienti a, b e c. Anche v_x, v_y e v_z sono espressioni composte.

In questo stadio siamo già in grado di decidere se la sfera viene intersecata o no dalla retta. Ciò viene determinato dalla discriminante $D = b^2 - 4*a*c$ (cioè dal termine sotto radice). Varrà:

$D<0$ => la retta non interseca la sfera
$D=0$ => la retta tocca la sfera in un punto
$D>0$ => la retta interseca la sfera in due punti

Ciò risulta dal fatto che una radice di un numero negativo ($D<0$) non è definita all'interno dei numeri reali. Se D é =0, anche la radice sarà uguale a zero, per cui tutto il radicale scompare. K sarà quindi uguale a -b/(2*a). Se D è >0, la radice sarà rilevante. Esistono due possibilità.

Il parametro determinato è la misura della distanza fra il punto di intersezione e l'osservatore. Normalmente il programma decide per il K più piccolo (cioè il punto di intersezione più vicino) sempre che non sia negativo. Infatti, se fosse negativo, si troverebbe dall'altra parte della lastra, per cui non sarebbe visibile. Se il K maggiore appartiene a una sfera, si tratterà del punto di intersezione tramite il quale il raggio esce dalla sfera. Da K è possibile calcolare in qualunque momento il punto esatto nello spazio, cosa che tuttavia, nella maggior parte dei casi, non è necessaria.

6.2.2. Superfici piane (Parallelogramma)

Il calcolo del punto di intersezione con una superficie piana a forma di parallelogramma (es. un rettangolo) è anch'esso fra i più semplici. I poligoni irregolari sono notevolmente più difficili da elaborare. Per parallelogramma si intende una superficie a quattro angoli, nella quale i lati che si trovano l'uno di fronte all'altro corrono paralleli. Se i lati si trovano ad angolo retto l'uno rispetto all'altro, si tratterà dell'eccezione chiamata rettangolo. Se inoltre tutti i lati hanno la stessa lunghezza, si tratterà di un quadrato.

Il principio è chiaro: dapprima viene calcolato un punto di intersezione (del raggio luminoso) con un piano. Quindi si dovrà verificare se il punto di intersezione si trova all'interno di determinati limiti.

Come definire un parallelogramma? Per la sfera abbiamo memorizzato il centro ed il raggio. Per una superficie è consigliabile fissare dapprima i parametri per l'equazione vettoriale di un piano:

$$P = P_0 + k^*\vec{v} + m^*\vec{w}$$

Il punto P_0 ed i vettori $\vec{v}$ e $\vec{w}$ devono anch'essi venire memorizzati. P_0 dovrebbe essere un punto angolare a piacere della superficie. Il vettore v avrà il piede in P_0 e la punta nel punto angolare più vicino. Il vettore $\vec{w}$ avrà anch'esso il piede in P_0, ma andrà all'altro punto angolare (ved. Fig. 6.5).

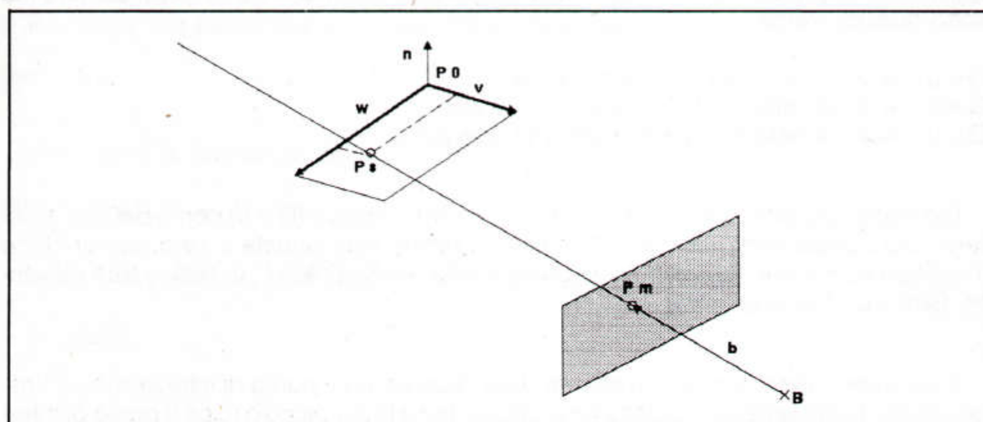


Fig. 6.5 Calcolo del punto di intersezione con un parallelogramma

Con queste premesse è molto rapido determinare se un punto si trova o no all'interno del parallelogramma. Se per il calcolo del punto in questione devono venire inseriti dei valori per K o per m, o per ambedue, maggiori di 1 o minori di 0, il punto si troverà al di fuori della superficie. Se al contrario vale:

$$0 \leqq k \leqq 1 \quad \text{e} \quad 0 \leqq m \leqq 1$$

il punto P si troverà all'interno del parallelogramma.

Passiamo ora al calcolo vero e proprio del punto di intersezione. In Fig. 6.5 è possibile ricostruire il tutto. Sia P_s il punto di intersezione ricercato. Esso si troverà sia sulla retta che sulla superficie. Per esso valgono quindi sia l'equazione per la retta che quella per la superficie:

$$P_s = P_m + s^*\vec{b} \quad \text{e}$$
$$P_s = P_0 + k^*\vec{v} + m^*\vec{w}$$

dove

P_s	punto di intersezione ?
P_m	punto sulla lastra
$\vec{b}$	vettore di direzione della retta
s	vettore di allungamento della retta
P_0	punto angolare del parallelogramma e piede dei vettori $\vec{v}$ e $\vec{w}$
$\vec{v},\vec{w}$	vettori di direzione sul piano
k,m	fattori di allungamento per $\vec{v}$ e $\vec{w}$.

A questo punto osserviamo di nuovo il capitolo 4.3. In esso troviamo una forma alternativa dell'equazione di una retta, l'equazione della normale e la formula per la trasformazione dell'una nell'altra. L'equazione della normale sarà quindi nel nostro caso:

$$(P_s-P_0) * \vec{n} = 0$$

$\vec{n}$ è la normale (il vettore perpendicolare al piano) e può venire determinata tramite il prodotto vettoriale di $\vec{v}$ e $\vec{w}$.

$$\vec{n} = \vec{v} \times \vec{w}$$

In seguito sarà molto più facile effettuare i calcoli in questo modo, anche se a tal scopo è necessario calcolare in parecedenza un prodotto vettoriale.

Utilizziamo quindi le equazioni delle rette in quelle per i piani:

$$(P_m + s*\vec{b} - P_0) * \vec{n} = 0 <=>$$
$$(P_m - P_0) * n + s*\vec{b}*\vec{n} = 0 <=>$$
$$s*\vec{b}*\vec{n} = (P_0-P_m)*\vec{n}$$

A questo punto è necessario effettuare una differenziazione tra i casi che si presentano. Infatti ora è possibile determinare se esiste o no un punto di intersezione.

Caso 1: $\vec{b}*\vec{n} = 0$:
In questo caso la parte di sinistra dell'equazione è zero:

$$0 = (P_0-P_m)*n$$

Nel caso in cui questa espressione sia vera, sarà possibile inserire qualunque valore a piacere per s, la retta giace sul piano. Esistono, per così dire, infiniti punti di intersezione.

Nel caso in cui l'espressione sia falsa, la retta correrà parallela al piano, quindi non esiste nessun punto di intersezione.

Caso 2: b*n <> 0

Qui possiamo andare avanti con i calcoli, ottenendo per s:

$$s = \frac{(P_0-P_m)*\vec{n}}{\vec{b}*\vec{n}}$$

Ma s indica il fattore di allungamento per il vettore di direzione della retta. Il punto di intersezione P_s può venire determinato.

A noi però interessa sapere se P_s non solo si trova sul piano, ma anche se si trova all'interno del parallellogramma. A tale scopo dobbiamo ricondurre la formula della normale dell'equazione del piano alla forma con K e m. I vettori $\vec{v}$ e $\vec{w}$ sono già noti:

$$P_s = P_0 + k*\vec{v} + m*\vec{w}$$

In forma di parametri:

$$p_{sx} = p_{0x} + k*v_x + m*w_x$$
$$p_{sy} = p_{0y} + k*v_y + m*w_y$$
$$p_{sz} = p_{0z} + k*v_z + m*w_z$$

Secondo la derivata di cui al cap. 4.3, scegliamo le due equazioni per le quali valga quanto segue (i e j sono al posto di x, y oppure di z):

$v_i <> 0$ e
$w_j*v_i - w_i*v_j <> 0$

(es. $v_x<>0$ e $w_z*v_x-w_x*v_z<>0$)

Se non esiste nessuna combinazione di due equazioni che adempiono alle condizioni, ci troviamo in presenza di un errore. I due vettori v e w non sono indipendenti linearmente.

Se si trovano le due equazioni che adempiono a queste condizioni (di solito sono le prime due) calcoleremo semplicemente K ed m come segue:

$$m = \frac{(p_{sj}-p_{0j})*v_i - (p_{si}-p_{0i})*v_j}{w_j*v_i - w_i*v_j}$$

$$k = (p_{si}-p_{0i} - m*w_i)/v_i$$

Prima del calcolo di K verifichiamo se m si trova all'interno dell'ambito permesso, che va da 0 a 1. Verifichiamo inoltre se anche k si trova in tale ambito. Se tutte queste condizioni sono adempiute, la retta inteseca il parallelogramma. Se s è invece negativo, il punto di intersezione si troverà dietro la lastra, per cui sarà invisibile.

Spesso i piani vengono posizionati in punti ben determinati, al fine di ridurre il tempo di calcolo. Di conseguenza, un pavimento potrebbe trovarsi per esempio nel piano xy con $z=0$. Le equazioni di cui sopra diverranno pertanto molto più semplici.

Una volta determinato che il raggio luminoso interseca la superficie, passiamo alle informazioni relative alla scelta del colore, in quanto sulla superficie potrà esserci una scritta o un piccolo disegno. Potremmo riuscirci tramite il fatto che ogni punto della superficie si trova in memoria sotto forma di Pixel binari, (cioè esattamente come in una normale memoria per immagini). Quindi i valori di K ed m indicano le coordinate del punto in questione sulla superficie. Sarà quindi possibile guardare in memoria cosa c'è in tale posizione. Potrà naturalmente trattarsi di informazioni su diverse caratteristiche della superficie. In tal modo sarà possibile "rispecchiare" la superficie in tali posizioni, con il suo disegno ecc.

A questo punto, siamo in grado di produrre e memorizzare un grafico o un testo con un normale programma di grafica. Questo grafico apparirà sull'immagine del Ray-Tracing sulla superficie corrispondente, ruotato o ingrandito. Spesso anche i motivi proiettati sulla superficie vengono determinati tramite un calcolo (es. scacchiera ecc.).

La procedura è eseguibile naturalmente anche per altri oggetti, quali sfere ecc., anche se con un dispendio maggiore per le singole proiezioni del grafico sull'oggetto.

6.3. Fonti di luce: Riflessioni, lucentezza, luci ed ombre

Prima di scrivere un programma di grafica il Ray-Tracing, continuiamo con questo capitolo: ne vale la pena. Infatti qui apprendiamo come realizzare gli influssi nel paesaggio della luce, degli oggetti a specchio, delle superfici lucide e opache nonché la loro formazione delle ombre. Tutti questi aspetti sono strettamente correlati fra loro. Lavoreremo con diverse leggi fisiche, che rivestono un ruolo importante per questi aspetti, e potremo anche riformulare leggi della natura, a nostro piacimento. Tali leggi, formule e deviazioni, tuttavia, non valgono per il Ray-Tracing. Sarà possibile incorporarle facilmente in altri algoritmi tridimensionali, producendo effetti sorprendenti.

E' chiaro che non è possibile tener conto, in questa sede, di tutti gli effetti. Ci occuperemo spesso di formule di approssimazione, che ci alleggeriranno sostanzialmente il lavoro. infatti sono molti gli aspetti che rivestono un ruolo importante ai fini del calcolo dei colori e dell'intensità della luce. Per esempio, tratteremo allo stesso modo tutti i colori, anche se sappiamo che nella realtà ogni lunghezza d'onda viene riflessa e rifratta in modo diverso dalle altre. Ogni materiale, sia esso carta, oro, argento o plastica, ha le proprie caratteristiche, non calcolabili ma deducibili solo da complicate tabelle. Non sarà tuttavia inutile dichiarare, per ogni singolo oggetto, se è costituito da carta bianca, vetro bianco ecc. Tutti questi dettagli essenziali verranno solo approssimati, al fine di ottenere l'immagine migliore possibile. Molti valori, costanti ecc., che verranno utilizzati in seguito, sono valori dedotti dalla prassi, e noi potremo, anzi dovremo, modificarli in qualunque momento.

6.3.1. Riflessione diffusa

Abbiamo sicuramente tutti sentito parlare della legge centrale di riflessione, che vale sia per la luce che per il gioco del biliardo: angolo di ingresso uguale all'angolo di uscita. Benché questa legge teorica valga naturalmente sempre, la realtà è molto più complessa. Se il raggio luminoso incontra una superficie, il modo e la direzione del riflesso della luce dipenderanno dalla natura di tale superficie. Per esempio una superficie lucida si atterrà, più o meno esattamente, alla legge della riflessione. Una superficie ruvida o opaca,invece, riflette il raggio luminoso in tutte le direzioni possibili. Più la superficie è opaca, maggiormente ampio sarà il riflesso del raggio luminoso. Una determinata parte di luce verrà distribuita addirittura in tutte le direzioni. Per tale parte di luce si parla di riflessione diffusa, in contrapposizione con la riflessione a specchio.

La Fig. 6.6 illustra questo comportamento. Per il riflesso diffuso, cioè quello che va in tutte le direzioni, l'angolo di incidenza del raggio luminoso sulla superficie ha importanza solo per la determinazione della luminosità del singolo punto. Più la luce arriva verticalmente, più luminoso sarà il punto. Sappiamo tutti che all'equatore la luce del sole arriva sulla terra quasi verticalmente, mentre da noi ha purtroppo un angolo inferiore, e che questo è il motivo per cui all'equatore è più caldo che qui.

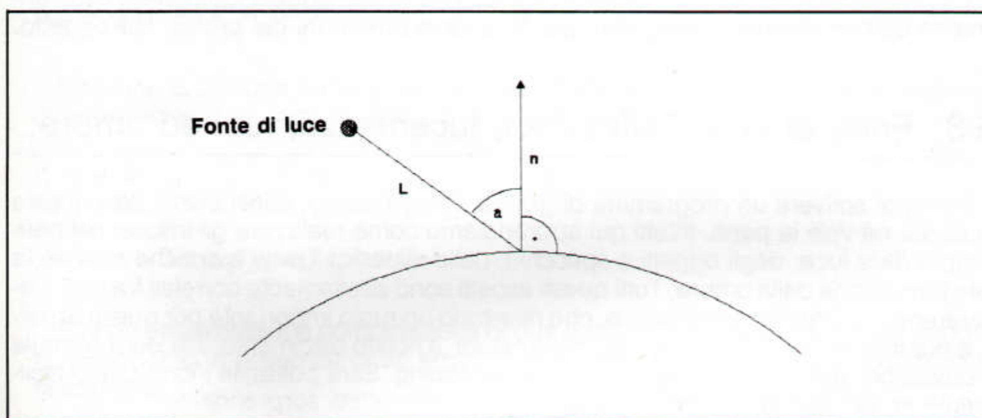


Fig. 6.6 Riflessione diffusa

Oltre alla fonte di luce dovremo anche tener conto della luce che cade su altri oggetti e viene riflessa su altri oggetti ancora e/o sul nostro punto. Immaginiamo quanto complicato sarebbe il calcolo. Prendiamo invece solo una luminosità diffusa di sottofondo, che viene comunque riflessa dal nostro punto. L'intensità luminosa che risulta da tale luminosità di sottofondo viene aggiunta alla luce che risulta dalla riflessione.

Giungiamo quindi alla seguente formula per l'intensità della luce irradiata da un punto:

$$I = I_h{*}K_{oh} + I_q{*}K_{od}{*}\cos(a)$$

e siccome

$$\vec{n} * \vec{L} = |\vec{n}| * |\vec{L}| * \cos(a) \quad <=>$$
$$\cos(a) = (n * \vec{L}) / (|\vec{n}| * |\vec{L}|) \quad <=>$$
$$\cos(a) = \vec{n}/|\vec{n}| * \vec{L}/|\vec{L}| \quad <=>$$
$$\cos(a) = \vec{n}' * \vec{L}'$$

varrà anche:

$$I = I_h * K_{oh} + I_q * K_{od} * \vec{n}' * \vec{L}'$$

dove:

I	intensità della luce irradiata dal punto
I_h	intensità della luminosità di sottofondo
K_{oh}	costante di riflessione per la riflessione dello sfondo con $0 \leqq K_{oh} \leqq 1$ (chiarezza dell'oggetto o del colore) (anche chiarezza o colore dell'oggetto)
I_q	intensità della fonte di luce
K_{od}	costante di riflessione per la riflessione diffusa con $0 \leqq K_{od} \leqq 1$ (chiarezza dell'oggetto o del colore) (anche chiarezza o colore dell'oggetto)
a	angolo fra il vettore della luce $\vec{L}$ e il vettore $\vec{n}$ normale al piano (= perpendicolare)
$\vec{L}$	vettore della luce
$\vec{L}'$	vettore unitario della luce (lunghezza = 1)
$\vec{n}$	vettore normale dal piano (perpendicolare)
$\vec{n}'$	vettore unitario normale al piano (lunghezza = 1)

L'unità per le intensità potrà venire da noi determinata arbitrariamente (es. 0-100 oppure 0-1). Le costanti K_{oh} e K_{od} dipendono dalle leggi di riflessione dell'oggetto in questione. Se l'oggetto riflette molto, i valori per K_{oh} e K_{od} si aggirano attorno a 1. Se riflette molto debolmente, questi valori saranno attorno a 0. Di solito questi due valori sono identici fra loro ($K_{oh} = K_{od}$). Essi forniscono una misura della luminosità di base dell'oggetto. Ogni oggetto assorbe una determinata porzione di luce. In presenza della stessa illuminazione, un oggetto risulta chiaro, un altro scuro.

Se un oggetto assorbe in ugual misura tutte le parti della luce fino ad una determinata porzione di essa (cioè, per noi, le parti di Rosso, Verde e Blu), in presenza di luce bianca esso apparirà grigio. Se assorbe (quasi) tutta la luce, sarà nero. Se invece (quasi) tutta la luce viene riflessa, esso ci apparirà bianco.

Alcuni oggetti assorbono solo determinate frequenze della luce. Se per esempio vengono assorbite tutte le parti di rosso e di blu, sarà solo il verde ad essere riflesso, e noi diremo che l'oggetto è verde.

Dal momento che avremo sempre a che fare con oggetti colorati, che possiedono delle intensità di base ben determinate per i colori rosso, verde e blu, le costanti K_{oh} e K_{od} rappresentano i valori di chiarezza per tali colori di base. Ogni colore di base ha il proprio valore di chiarezza. La formula vista in precedenza dovrà quindi venire calcolata per ciascuno dei tre colori di base. In questo modo saranno possibili anche fonti di luce colorata, dal momento che la fonte di luce può avere anch'essa diversi valori di intensità I_q per le tre porzioni di colore. Il risultato I sarà quindi il valore della chiarezza del punto per il singolo colore di base, es. verde.

Finora non si è tenuto conto di un fattore: la distanza. Maggiore è il percorso che la luce ripercorre, più debole essa diverrà, in quanto si sparge. Normalmente l'intensità della luce diminuisce proporzionalmente al quadrato della distanza, cioè $I' = I/d^2$, dove d è il valore della distanza. Per adeguarsi all'estetica dell'immagine si è determinata la formula $I' = I/(d * K_d)$, dove K_d è una costante arbitraria. Per semplicità viene sempre misurata la distanza fra la fonte di luce e il punto dell'oggetto, anche se dovrebbe venire considerata anche la distanza rispetto all'osservatore. Applicandola alla porzione di fonte di luce della nostra formula, avremmo:

$$I = I_h * K_{oh} + \frac{I_q * K_{od} * \vec{n}' * \vec{L}'}{d * K_d}$$

6.3.2. Riflessioni a specchio

Con ciò abbiamo a disposizione un modello molto semplice di luminosità per il nostro programma di Ray-Tracing. Possiamo tuttavia andare avanti fino al riflesso a specchio. In questo caso con la parola "specchio" non intendiamo che tutti gli oggetti si riflettano nettamente sull'oggetto "a specchio", cosa che accadrebbe se l'angolo di incidenza fosse uguale a quello di uscita (in altre parole: per un riflesso a specchio perfetto l'angolo b deve essere sempre uguale a 0). I raggi di luce in ingresso possono invece venire riflessi all'interno di un determinato campo angolare (ved. Fig. 6.7). L'intensità del riflesso cala con la distanza del vettore dello sguardo $\vec{S}$ dal raggio esatto del riflesso $\vec{R}$. Quindi, maggiore è l'angolo fra $\vec{R}$ ed $\vec{S}$, minore è l'intensità di riflessione, calcolabile con la seguente formula:

$$I_s = I_q * w(i,l) * \cos(b)^f$$

e siccome

$$\vec{R} * \vec{S} = |\vec{R}| * |\vec{S}| * \cos(b) \quad \Leftrightarrow$$
$$\cos(b) = (\vec{R} * \vec{S}) / (|\vec{R}| * |\vec{S}|) \quad \Leftrightarrow$$
$$\cos(b) = \vec{R}' * \vec{S}'$$

vale anche

$$I_s = I_q * r(a,l) * (\vec{R}' * \vec{S}')^f$$

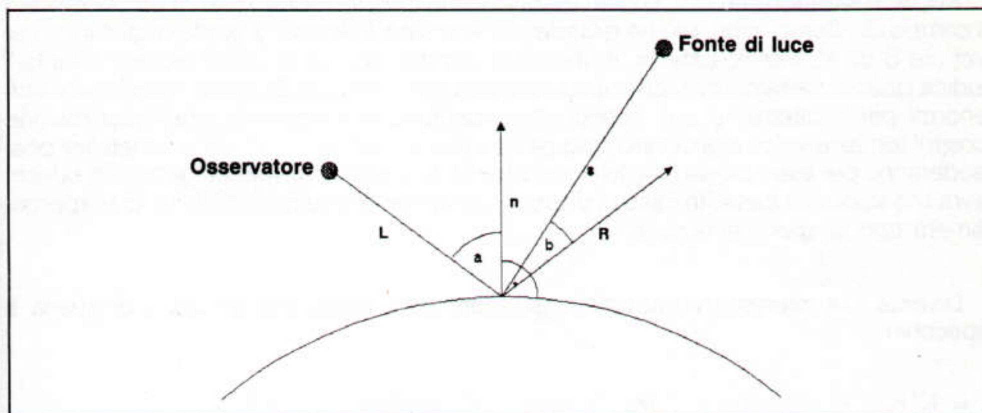


Fig. 6.7 Riflessione a specchio

dove:

I_s intensità del punto tramite il riflesso a specchio
I_q intensità della fonte di luce
b angolo fra il vettore esatto del riflesso $\vec{R}$ e quello dello sguardo $\vec{S}$
R vettore esatto del riflesso
R' vettore unitario del riflesso (lunghezza = 1)
$\vec{S}$ vettore dello sguardo (Punto -> osservatore). Attenzione al fatto che, precedentemente, avevamo orientato il vettore dello sguardo dall'osservatore al punto. In questo caso sarà necessario ricalcolarlo ($\vec{S} = -\vec{s}$).
$\vec{S}$' vettore unitario dello sguardo (Lunghezza = 1)
r(a,l) funzione di riflessione a seconda dell'angolo di incidenza a e della lunghezza d'onda della luce l con $0 \leq r(a,l) \leq 1$.
f messa a fuoco

In questo caso non si è ancora tenuto conto della distanza dalla fonte di luce. La funzione di riflessione r(a,l) dipende completamente dal materiale. I diversi materiali quali metallo, plastica ecc. possiedono infatti un grado di riflessione molto diverso, a seconda di quali colori vengono irradiati e quindi da quali lunghezze d'onda sia composta la luce. Ciò significa che i materiali modificano molto leggermente la luce che si specchia su di essi. E' per lo stesso motivo che per esempio molti specchi non sono completamente fedeli nella riproduzione dei colori. Secondariamente il grado di riflessione dipende, in maniera diversa a seconda di diversi materiali, dall'angolo di incidenza. Tuttavia, non vogliamo occuparci in questa sede di tale complicata funzione, ma la sostituiremo, per approssimazione, con una costante K_r, che potrà venire scelta a seconda delle necessità:

$r(a,l) = K_r \qquad \text{con } 0 \leq K_r \leq 1$

Anche la variabile f è una costante del materiale. Essa indica quanto è concentrato il campo di riflesso, cioè, più f è grande, minore sarà il riflesso, a parità di distanza del vettore $\vec{S}$ da $\vec{R}$. Per $\vec{R}$, tuttavia, il riflesso è sempre intenso in ugual misura. Quindi f indica quanto nettamente viene riprodotta una fonte di luce. Se impostiamo dei valori enormi per f, otterremo una specchiatura perfetta, dal momento che l'espressione $\cos(b)^f$ fornisce valori significativi solo per $b = 0 => \cos(b) = 1$. Gli oggetti metallici possiederanno per esempio un f molto elevato (f = 50 fino a f = 100), mentre gli oggetti opachi avranno valori più bassi. In caso di dubbio, come per tutte le altre costanti, ci si approssimerà con la sperimentazione.

Diventa ora interessante abbinare gli effetti della riflessione diffusa e di quella a specchio:

$$I = I_h * K_{oh} + \frac{I_q}{d * K_d} * [K_{od} * \cos(a) + K_r * \cos(b)^f] \quad <=>$$

$$I = I_h * K_{oh} + \frac{I_q}{d * K_d} * [K_{od} * (n' * L') + K_r * (R' * S')^f]$$

I singoli parametri dovrebbero esserci già noti. Le sostanti K_{oh}, K_d, K_{od}, K_r ed f offrono spazio per le nostre sperimentazioni. I parametri I_h e I_q sono costanti delle fonti di luce. Per queste formule si dovrà calcolare d, a e b, mentre le espressioni cos(a) e cos(b) sono sostituibili, come abbiamo già visto, con i prodotti scalari dei vettori unitari corrispondenti (un vettore unitario viene calcolato tramite il quoziente di vettore e lunghezza del vettore).

Le formule indicate si riferiscono esclusivamente ad una sola fonte di luce. Se ci troviamo in presenza di più fonti di luce, le singole intensità luminose verranno sommate l'una all'altra (tuttavia si potrà usare l'espressione $I_h * K_{oh}$ solo una volta, in quanto essa determina la luminosità di sottofondo).

6.3.3. Calcolo del raggio di riflessione

In quanto visto precedentemente abbiamo sempre ipotizzato che la direzione del vettore di riflessione $\vec{R}$ ci fosse nota. In caso di riflessione a specchio e totale questo vettore dovrà invece essere calcolato. Ved. a tal scopo lo schema allegato (fig. 6.8).

Si ipotizza di conoscere il vettore normale $\vec{n}$ nel punto P, dove si specchia il raggio di luce $\vec{L}$. Il vettore normale è, com'è noto, quello che cade perpendicolarmente sulla superficie in questione di un oggetto. Il suo calcolo varia da superficie a superficie (in caso di sfera esso sarà semplicemente il vettore del raggio al punto di specchiatura P, in caso di piano esso sarà il prodotto vettoriale di due vettori indipendenti linearmente e giacenti su quel piano). La lunghezza di n in questo caso è trascurabile. Quindi sono noti $\vec{L}$ (il vettore che va o proviene dalla fonte di luce specchiantesi, il suo verso è ininfluente) ed $\vec{n}$.

Come visibile nello schizzo, il vettore di riflessione $\vec{R}$ viene calcolato da:

$\vec{R} = -\vec{L} + 2*\vec{L}'$

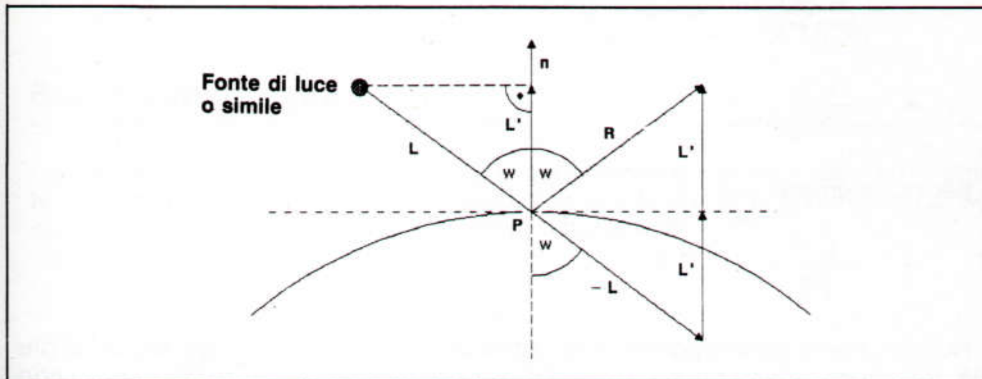


Fig. 6.8 Calcolo del vettore di riflessione

$\vec{L}'$ sarà quel vettore che deriva dalla proiezione di $\vec{L}$ su $\vec{n}$. La sua determinazione è solo un po' più complessa. $\vec{L}'$ giace quindi su $\vec{n}$, per cui è solo un allungamento o un accorciamento di $\vec{n}$:

$\vec{L}' = k * \vec{n}$

Il fattore di allungamento K è l'incognita che dovremo cercare. In figura riconosceremo un triangolo rettangolo con i lati $|\vec{L}|$ (ipotenusa) ed $|\vec{L}'|$ (cateti) e con l'angolo w fra tali lati. Conformemente alla definizione del coseno (ved. appendice) varrà:

$\cos(w) = |\vec{L}'| / |\vec{L}|$ <=>

$|\vec{L}'| = |\vec{L}| * \cos(w)$

Utilizziamo quindi per $\vec{L}'$ l'equazione di cui sopra:

$|k*\vec{n}| = |\vec{L}|* \cos(w)$ <=>

$k*|\vec{n}| = |\vec{L}|* \cos(w)$ <=>

$k = |\vec{L}|/|\vec{n}|* \cos(w)$

Bene. Ora abbiamo una formula per K. Tuttavia l'angolo w, o il suo coseno cos(w) non ci è noto. Ed ecco che ci aiuta il prodotto scalare di due vettori:

$\vec{L}*\vec{n} = |\vec{L}|*|\vec{n}|*\cos(w)$ <=>

$$\cos(w) = \frac{\vec{L}*\vec{n}}{|\vec{L}|*|\vec{n}|}$$

Naturalmente useremo immediatamente per cos(w) questa espressione, i cui componenti sono tutti noti:

$$k = \frac{|\vec{L}|*\vec{L}*\vec{n}}{|\vec{n}|*|\vec{L}|*|\vec{n}|} \quad <=>$$

$$k = \frac{\vec{L}*\vec{n}}{|\vec{n}|^2}$$

per cui otterremo per $\vec{L}$':

$$L' = \frac{\vec{L}*\vec{n}}{|\vec{n}|^2} * \vec{n}$$

Non cerchiamo però di portare n nel numeratore, rendendolo $|\vec{n}|^2$, per ragioni di brevità. Ricordiamo infatti che, nella moltiplicazione scalare, la sequenza deve venire rigorosamente mantenuta; ad esempio, $(\vec{a}*\vec{b})*\vec{c} = \vec{a}*(\vec{b}*\vec{c})$ non vale.

6.3.4. Formazione di ombre

Il tener conto delle ombre, prodotte da corpi su altri oggetti, è normalmente molto simile al problema delle superfici nascoste. In pratica abbiamo delle superfici che nascondono altre superfici rispetto alla fonte di luce esattamente come rispetto all'osservatore. Quando si tratta di più di un oggetto, le soluzioni saranno proporzionalmente complicate. Nel caso del Ray-Tracing, invece, la loro gestione è un gioco da ragazzi e non vale quasi che vi si dedichi un capitolo.

Il principio viene enunciato molto velocemente: ipotizziamo di avere calcolato il punto di intersezione del raggio luminoso con un oggetto. Formiamo ora un secondo raggio da tale punto alla fonte di luce (o alle fonti di luce). Con tale raggio procederemo esattamente come con il raggio dello sguardo. Calcoliamo i punti di intersezione di questa retta con tutti gli altri oggetti. In questa sede è necessario verificare solo se esiste almeno un oggetto incontrato dal raggio. Se esiste effettivamente un punto di intersezione con un altro oggetto, esso coprirà la fonte di luce; il punto di cui si vuole determinare l'intensità si trova in ombra.

Esso potrebbe essere illuminato al massimo da altre fonti di luce, infatti qui non si è tenuto conto del fatto che un oggetto possa anche riflettere, cioè irradiare sul punto la luce di un'altra fonte di luce. Abbiamo tralasciato per semplicità questa considerazione, che tuttavia è tenuta in conto in complicati programmi professionali.

Questo punto sarebbe quindi normalmente nero. Dal momento però che esiste una luminosità di sottofondo, esso riceverà nonostante tutto una determinata intensità. Se un punto si trova in ombra rispetto ad una fonte di luce, l'intera espressione per tale fonte di luce cui eravamo pervenuti nel capitolo precedente, decade, cioè diventa zero. Ciò che resta è la luminosità di sottofondo e la luce di un'altra eventuale fonte di luce.

Il fatto che un oggetto possa schermare altri oggetti da una fonte di luce potrà venire utilizzato quando applicheremo, ad uno o più lati della fonte di luce, dei piani non trasparenti, che delimitino il raggio di luce. Con ciò si potrebbero simulare per esempio dei fari direzionali, che illuminano solo zone ben precise di una stanza o di un paesaggio.

6.4. Il programma

Finalmente siamo al termine della grigia teoria incontrata fin qui. Possiamo finalmente mettere in pratica direttamente sullo schermo quanto abbiamo appreso finora. Sipario sul famoso programma di Ray-Tracing, naturalmente in C.

```
/**************************************/
/**                                 **/
/**       Computer e realta'        **/
/**                                 **/
/**         3-D-RAY-TRACING         **/
/**            32 Colori            **/
/**               con               **/
/**       diverse fonti Luce,       **/
/**      Riflessione diffusa e      **/
/**           a specchio,           **/
/**        Riflessione totale       **/
/**      e formazione di onde       **/
/**                                 **/
/**         Organizzazione:         **/
/**       Modello volumetrico       **/
/**     orientato all'oggetto       **/
/**                                 **/
/**           Axel Plenge           **/
/**                                 **/
/**                                 **/
/**                                 **/
/**************************************/


#include <exec/types.h>
#include <intuition/intuition.h>
#include <libraries/mathffp.h>

/* Dichiarazione delle funzioni (solo per Compilatore Aztec): */
/* a scelta anche: #include <functions.h>                      */
/************************************************************/
VOID                    ClearScreen();
VOID                    Close();
VOID                    Exit();
struct Message *        GetMsg();
UWORD                   GetRGB4();
LONG                    IoErr();
```

```
VOID                    Move();
struct FileHandle *     Open();
struct Library *        OpenLibrary();
struct Screen *         OpenScreen();
struct Window *         OpenWindow();
VOID                    ReplyMsg();
VOID                    SetAPen();
VOID                    SetRGB4();
LONG                    Wait();
LONG                    Write();
VOID                    WritePixel();
/**************************************************/

#define WURZEL SPSqrt
#define SIN    SPSin
#define COS    SPCos
#define TAN    SPTan
#define LOG    SPLog
#define EXP    SPExp

#define MODE_NEWFILE 1006L

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
LONG   *MathBase;
LONG   *MathTransBase;

/* Costanti di hardware: */

#define  ANZ_FARBEN  32          /* Numero dei diversi colori     */
#define  ANZ_EBENEN  5           /* Numero dei livelli di colore */
#define  ANZ_INT     16          /* Numero livelli di intensita' */
#define  X_AUFL      320         /* Risoluzione x dell'Hardware  */
#define  Y_AUFL      256         /* Risoluzione y dell'Hardware  */
#define  G_MOD       NULL        /* Modo grafico                 */
#define  ANZ_PAL_FAB 32          /* Numero registri di tavolozza */

extern ULONG palette[ANZ_FARBEN][3];

/* Inizializzazione di una struttura per un nuovo schermo */
/*********************************************************/

struct NewScreen NeuerBildschirm =
{
   0,                    /* Coordinata x superiore sinistra  */
                         /* Angolo (sempre 0)                */
   0,                    /* Coordinata y superiore sinistra  */
                         /* Angolo                           */
   X_AUFL,               /* Larghezza schermo                */
   Y_AUFL,               /* Altezza schermo                  */
```

```
    ANZ_EBENEN,              /* Numero piani di immagine        */
    32,                      /* Colore dei dettagli             */
    1,                       /* Colore delle superfici          */
    G_MOD,                   /* Modo grafico                    */
    CUSTOMSCREEN,            /* Tipo di schermo                 */
    NULL,                    /* nessun nuovo Font               */
    "Ray-Tracing-Demo",      /* Testo nell'intest. dello schermo */
    NULL,                    /* non usato, sempre ZERO (NULL)   */
    NULL,                    /* nessun BitMap proprio           */
};

/* Inizializzazione struttura per nuova finestra: */
/**************************************************/

struct NewWindow NeuesFenster =
{
    0,                       /* Coordinata x ang. sup. sin.     */
    10,                      /* Coordinata y ang. sup. sin.     */
    X_AUFL,                  /* Larghezza finestra              */
    Y_AUFL-10,               /* Altezza finestra                */
    32,                      /* Colore dei dettagli             */
    1,                       /* Colore delle superfici          */
    MOUSEBUTTONS |           /* Segnalazione per tasto del Mouse */
     RAWKEY,                 /* e tasto                         */
    ACTIVATE |               /* Selezione elementi e tipi       */
     BORDERLESS|             /* della finestra: senza bordi     */
     NOCAREREFRESH |         /* senza segnalazioni di Refresh   */
     RMBTRAP,                /* senza operazioni di Menu        */
    NULL,                    /* senza gadget propri             */
    NULL,                    /* CheckMark                       */
    NULL,                    /* Testo di intestazione finestra  */
    0,                       /* Indirizzo Struttura di schermo  */
                             /* deve venire inizializzato       */
                             /* entro il programma, dopo        */
                             /* l'apertura di uno               */
                             /* schermo                         */
    NULL,                    /* Nessuna Finestra di SuperBitmap */
    X_AUFL,                  /* Larghezza minima                */
    Y_AUFL-10,               /* Altezza minima                  */
    X_AUFL,                  /* Larghezza massima               */
    Y_AUFL,                  /* Altezza massima                 */
    CUSTOMSCREEN,            /* Tipo schermo                    */
};

struct Screen *Bildschirm;
struct Window *Fenster;
struct RastPort *RastPort;
struct IntuiMessage *Message;

VOID main()
{
 LONG do_program();
```

```
LONG fehler;           /* Per le abbreviazioni dei tipi, ved. */
                       /* i file di Include di Exec: types.h   */

/* Apertura delle library di Intuition, Grafiche, e Mathe  */

printf("Ray-Tracing / C-Demo\n");

IntuitionBase =
(struct IntuitionBase *)OpenLibrary("intuition.library",0L);
if (IntuitionBase == NULL)                fehler = 1;
else
{
 GfxBase =
 (struct GfxBase *)OpenLibrary("graphics.library",0L);
 if (GfxBase == NULL)                     fehler = 2;
 else
 {
  MathBase =
  (LONG *)OpenLibrary("mathffp.library",0L);
  if (MathBase == NULL)                   fehler = 3;
  else
  {
   MathTransBase =
   (LONG *)OpenLibrary("mathtrans.library", 0L);
   if (MathTransBase == NULL)
   {
     printf("nessuna Library  MathTrans disponibile!\n");
     fehler = 4;
   }
   else
   {
    /* Apertura schermo:   */
    Bildschirm =
    (struct Screen *)OpenScreen(&NeuerBildschirm);
    if (Bildschirm == NULL)               fehler = 5;
    else
    {
     /* Apertura finestra:  */
     /* Aggiungere indirizzo della struttura di schermo:   */
     NeuesFenster.Screen = Bildschirm;
     Fenster =
     (struct Window *)OpenWindow(&NeuesFenster);
     if (Fenster == NULL)                 fehler = 6;
     else
     {
      RastPort = Fenster->RPort; /* RastPort merken       */

      Move(RastPort, 0L, 0L);
      ClearScreen(RastPort);     /* Cancellazione schermo */
      fehler = do_program();     /* chiamata del programma */
                                 /* vero e proprio         */

      CloseWindow(Fenster);      /* Chiusura finestra      */
     }
```

```
      CloseScreen(Bildschirm);      /* Chiusura schermo         */
     }
     CloseLibrary(MathTransBase); /* Chiusura Mathe-Library */
    }
    CloseLibrary(MathBase);       /* Mathe-Library close    */
   }
   CloseLibrary(GfxBase);         /* Graphics-Lib. close    */
  }
  CloseLibrary(IntuitionBase);   /* Intuition-Lib. close   */
 }
 Exit(fehler);                   /* Uscita con cod. err.   */
}


/************************/
/* Programma Principale */
/************************/

/* Defines e Macro: */

/* Determinazione del massimo/minimo di due valori: */
#define MAX(a,b)   ( (a) > (b) ? (a) : (b) )
#define MIN(a,b)   ( (a) < (b) ? (a) : (b) )

/* Determinazione del valore assoluto di un numero: */
#define ABS(a) ( (a) < 0  ?  -(a) : (a) )

/* Determinazione della Potenza a^b:    */
/* nach p=a^b <=> p=e^(b*ln(a)) */
#define POTENZ(a,b) (EXP( (b)*LOG(a) ))

/* Operazioni coi vettori come Macro:    */
/* a, b, c devono essere nomi di vettori */

/* Attribuzione di un vettore: a=b: */
#define VEKLET(a,b) {a[0]=(b[0]); a[1]=(b[1]); a[2]=(b[2]);}

/* Attribuzione di una matrice di colori: a=b */
#define COLLET  VEKLET

/* Somma vettoriale di due vettori c=a+b  */
#define VEKADD(c,a,b) {c[0] = (a[0]) + (b[0]);\
                       c[1] = (a[1]) + (b[1]);\
                       c[2] = (a[2]) + (b[2]);}

/* Sottrazione vettoriale di due vettori c=a-b */
#define VEKSUB(c,a,b) {c[0] = (a[0]) - (b[0]);\
                       c[1] = (a[1]) - (b[1]);\
                       c[2] = (a[2]) - (b[2]);}

/* Moltiplicazione b=z*a di un vettore per un numero: */
#define MULVEK(b,z,a) {b[0] = (z) * (a[0]);\
                       b[1] = (z) * (a[1]);\
                       b[2] = (z) * (a[2]);}
```

```
/* Prodotto scalare a*b di due vettori */
#define SKAPRD(a,b) ( (a[0])*(b[0]) +\
                      (a[1])*(b[1]) +\
                      (a[2])*(b[2]) )

/* Prodotto vettoriale c=axb di due vettori: */
#define VEKPRD(c,a,b) {c[0] = (a[1])*(b[2]) - (a[2])*(b[1]);\
                       c[1] = (a[2])*(b[0]) - (a[0])*(b[2]);\
                       c[2] = (a[0])*(b[1]) - (a[1])*(b[0]);}

/* Determinazione dell'equazione di una retta P = P0 + s*a   */
/* nella struttura struct gerade g (strutt. retta)           */
/* da due pinti b e c: P0=b, a=b-c                           */
/* Con cio' la vetta del vettore di direzione                */
/* e' contemporaneamente punto fisso della retta             */

#define GET_GERADE(g,b,c) {VEKLET(g.p0,b); VEKSUB(g.a,b,c);}

/* Calcolo di un punto                              */
/* dalla equazione vettoriale di una retta          */
/* P = P0 + s*a                                     */

#define GERADEN_PKT(P,P0,s,a) {P[0] = (P0[0]) + (s)*(a[0]);\
                               P[1] = (P0[1]) + (s)*(a[1]);\
                               P[2] = (P0[2]) + (s)*(a[2]);}

/* Calcolo della lunghezza di un vettore a:         */
#define VEK_LAENGE(a) (WURZEL(SKAPRD(a,a)))

/* Caratteristiche dell'oggetto (tipi): */
#define HINTERGRUND  0
#define LICHTQUELLE  1
#define KUGEL        2
#define FLAECHE      3

/* Tipi di operazioni della luce:       */
#define NORMAL       0
#define VERSPIEGELT  1
#define DURCHSICHTIG 2

/* Altri parametri: */
#define REKUR_MAX     7        /* N.ro Max. di Recursioni */
#define UNENDLICH     1.0e10   /* Lontano                 */
#define SCHWARZ       0.10     /* Limite della luce       */
#define MINI_S        0.0001   /* s minimo permesso       */
                               /* (quasi 0 a causa della  */
                               /*  imprecis. di calcolo)  */

/* Variabili globali e strutture: */
/**********************************/
```

```
/* Punto singolo di un oggetto:   */
struct punkt
{
   FLOAT    pos[3];          /* Posizione del Punto           */
   FLOAT    s;               /* valore s per equaz. retta     */
   FLOAT    farbe[3];        /* Colore                        */
   FLOAT    intens_b[3];     /* Valore intensita' RGB         */
   FLOAT    user0[3];        /* altri valori                  */
   FLOAT    user1;
   FLOAT    user2;
};

/* Parametri retta secondo P = P0 + s*a: */
struct gerade
{
   FLOAT    p0[3];           /* Punto fisso P0                */
   FLOAT    a[3];            /* Vettore di direzione a        */
};

/* Parametri del piano secondo P = P0 + s*a + t*b: */
struct ebene
{
   FLOAT    p0[3];           /* Punto fisso P0                */
   FLOAT    a[3];            /* Vettore di direzione a        */
   FLOAT    b[3];            /* Vettore di direzione b        */
   BOOL     n_valid;         /* Flag: TRUE => Vettore         */
                             /* della normale n disponibile */
                             /*       FALSE => n non e'       */
                             /*       disponibile             */
   FLOAT    n[3];            /* Vettore normale del piano     */
};

/* Costanti di materiale per un Oggetto: */
struct material
{
   FLOAT    farbe[3];        /* Colore RGB dell'oggetto       */
   FLOAT    farbe2[3];       /* eventuale secondo colore      */
   FLOAT    farbe3[3];       /* eventuale terzo colore        */
                             /* assumono il ruolo delle       */
                             /* Costanti Koh e Kod            */
   FLOAT    spieg_ref[3];    /* Costante per la riflessione */
                             /* specchiante Kr                */
   FLOAT    fokus_ref[3];    /* Messa a fuoco della rifles- */
                             /* sione a specchio f            */
   FLOAT    spiegel_int[3];/* Intensita' della luce         */
                             /* specchiata                    */
   BYTE     licht_typ;       /* Tipo operazione luce          */
                             /* =0: Normale                   */
                             /* =1: Totalmente specchiata     */
                             /* =2: trasparente               */
   BYTE     oberf_typ;       /* Tipo superficie               */
                             /* =0: Normale, tinta unita      */
                             /* =1: esempio: riquadri         */
                             /* =2: esempio: con motivi       */
   union                     /* Parametro individuale         */
   {
```

```
      struct
      {
         WORD anz_k,anz_m; /* Numero motivi in direz. k/m.   */
         WORD muster[16];  /* Definizione del motivo         */
      } muster;
      struct
      {                    /* oppure integro                 */
         WORD user1[3];
         WORD user2[3];
         WORD user3[3];
      } int_user;
      struct
      {                    /* oppure float                   */
         FLOAT user1[3];
         FLOAT user2[3];
         FLOAT user3[3];
      } float_user;
   } user;
};

/* Sfondo "infinito"  */
struct hintergrund
{
   WORD     typ;           /* Tipo per sfondo: 0             */
   FLOAT    unendlich;     /* Valore per la posizione        */
   FLOAT    farbe1[3];     /* Colore 1                       */
   FLOAT    farbe2[3];     /* Colore 2                       */
};

/* Fonte di luce: */
struct lichtquelle
{
   WORD     typ;           /* Tipo per fonte di luce: 1      */
   FLOAT    pos[3];        /* Posizione della fonte di luce */
   FLOAT    radius;        /* Raggio                         */
   FLOAT    farbe[3];      /* Colore della luce              */
   FLOAT    dist_konst;    /* Costante distanza Kd           */
};

/* Sfera: */
struct kugel
{
   WORD     typ;           /* Tipo per sfera: 2              */
   FLOAT    pos[3];        /* Posizione centro               */
   FLOAT    radius;        /* Raggio                         */
   struct material material; /* Costanti materiale           */
};

/* Superficie (Parallelogramma):  */
struct flaeche
{
   WORD     typ;           /* Tipo per superficie: 3         */
   FLOAT    p1[3],p2[3],   /* tre angoli                     */
            p3[3];
```

```
   struct ebene ebene;        /* Parametri del piano con       */
                              /* Vettore normale!              */
   struct material material; /* Costanti materiale             */
};

/* Osservatore: */
struct beobachter
{
   FLOAT     pos[3];          /* Posizione dell'osservatore    */
   FLOAT     blickp[3];       /* Punto al quale sta guardando  */
   FLOAT     blickvek[3];     /* alternativa: vettore vista    */
                              /* se il vettore vista e' 0      */
                              /* verra' utilizzato il punto    */
                              /* dello sguardo                 */
   FLOAT     mattsch;         /* Distanza lastra in vetro      */
   FLOAT     x_punktgr;       /* Larghezza punto               */
   FLOAT     y_punktgr;       /* Altezza ounto                 */
   LONG      x_aufl;          /* Risoluzione x                 */
   LONG      y_aufl;          /* Risoluzione y                 */
   FLOAT     x_winkel;        /* Angolo dello sguardo orizz.   */

   /* Variabili da calcolare in seguito: */
   FLOAT     blickv[3];       /* Vettore sguardo con lunghezza */
                              /* Osservatore -> lastra vetro   */
   FLOAT     x_step[3];       /* Vettore ampiezza passo z      */
   FLOAT     y_step[3];       /* Vettore ampiezza passo y      */
};

/* Tutto il mondo: */
struct welt
{
   WORD      anz_obj;            /* Numero di tutti gli ogg.    */
   WORD      anz_kugeln;         /* Numero di tutte le sfere    */
   struct kugel *kugeln;         /* Tutte le sfere              */
   WORD      anz_flaech;         /* Numero di tutte le superf.  */
   struct flaeche *flaech;       /* Tutte superfici             */
   WORD      anz_licht;          /* Numero di tutte le luci     */
   struct lichtquelle *licht;    /* Tutte le fonti di luce      */
   WORD      anz_hinten;         /* Numero degli sfondi = 1     */
   struct hintergrund *hinten;/* ampiezza, il Nulla             */
   FLOAT     hintlicht[3];       /* Luce di sottofondo          */
};

/* Variabili Globali: */

FLOAT     sch_p[3];          /* Intersezione attuale          */
FLOAT     sch_m, sch_k;      /* Fattore di allungamento piano */
WORD      *z_objekt;         /* Variabile interm. per oggetto */
WORD      rekur_zaehl;       /* Contatore recursivo           */

/* Definizione codici tasti RAW: */
#define ESC         0x45
#define TAS_S       0x21
```

```
LONG do_program()
{
   VOID      init_welt(),
             init_farben(),
             get_konst(),
             get_punkt(),
             ray_trace(),
             plot(),
             save_iff();
   USHORT    taste();

   USHORT code = 0;

   struct punkt      punkt;  /* Punto oggetto               */
   struct gerade     strahl; /* Raggio luminoso             */
   struct welt       welt;   /* Struttura mondo             */
   struct beobachter beo;    /* Struttura osservatore       */

   LONG      xa, ya;         /* mezze risoluzioni           */
   REGISTER LONG x, y;       /* indici di loop              */
   FLOAT     akt_punkt[3];   /* Punto sulla lastra in vetro */


   init_welt(&welt, &beo);   /* Inizializz. strutture dati  */

   init_farben();            /* Inizializz. tavolozza col.  */

   /* Calcolo ampiezza passi e vettore sguardo: */
   get_konst(&beo);

   rekur_zaehl = 0;          /* Contatore recursivo a zero  */


   /* Loop principali: */
   /********************/

   for (y = -(ya = beo.y_aufl/2); y<=ya && code!=ESC; y++)
   {
      for (x = -(xa = beo.x_aufl/2); x<=xa && code!=ESC; x++)
      {
         /* Calcolo punto attuale su lastra: */
         get_punkt(&beo, x, y, &akt_punkt[0]);

         /* Determinaz. equazione retta del raggio dello sguardo */
         /* con il punto sulla lastra come punto fisso           */
         GET_GERADE(strahl, akt_punkt, beo.pos);

         punkt.intens_b[0] =  /* Inizializz. coeff. intensita'    */
         punkt.intens_b[1] =
         punkt.intens_b[2] = 1.0;
         punkt.farbe[0] =     /* Impost. a zero val. col. punto   */
         punkt.farbe[1] =
         punkt.farbe[2] = 0.0;
```

```
        /* Qui inizia il RayTracing (inseguimento raggio) */
        /* Determinazione del colore del punto:           */
        ray_trace(&welt, &strahl, &punkt);

        /* Traccia il punto nel colore determinato: */
        plot(xa+x, ya-y, &punkt);

        /* Controllo tasto */
        code = taste();
     }
  }
  /* Attesa di ESC: */
  while (code != ESC)
  {
     /* Attesa messaggio: */
     Wait(1L << Fenster->UserPort->mp_SigBit);
     code = taste();

     if (code == TAS_S)  /* con "s":                      */
        save_iff();      /* memorizz.immag. in formato IFF */
  }

  return ((LONG)TRUE);
}


/* Controllo tasto: */
USHORT taste()

{
  ULONG  class;               /* Memoria per la struttura   */
  USHORT code, qualifier;     /* di Messaggio di Intuition  */
  APTR   address;
  SHORT  mouse_x, mouse_y;

  code = 0;

  /* Elaborazione segnalazioni: */
  /*******************************/
  while (code != ESC  &&
         (Message =
           (struct IntuiMessage *)GetMsg(Fenster->UserPort)) )
  {
     /* Lettura dati dalla struttura di messaggio: */
     class     = Message->Class;
     code      = Message->Code;
     qualifier = Message->Qualifier;
     address   = Message->IAddress;
     mouse_x   = Message->MouseX;
     mouse_y   = Message->MouseY;
     ReplyMsg(Message);    /* Restituzione Messaggio */
  }
  return (code);
```

```
/* Memorizzazione intera immagine in formato IFF */
/* sotto il nome: "Ray_Trace.pic" nella          */
/* directory attuale                             */
/* Attenzione! Un eventuale file gia' presente   */
/* e avente lo stesso nome verra'                */
/* riscritto!!!                                  */

VOID save_iff()

{
   LONG      i, row;
   UWORD     anz_farben;
   UWORD     farbe;
   ULONG     laenge;
   LONG      longword;         /* Memoria dati                       */
   WORD      word;
   BYTE      byte;
   BYTE      *BitPlanes[8];    /* Puntat. ad un max. di 8 Bitplanes */

   struct BitMap *BitMap;
   struct ColorMap *ColorMap;
   struct FileHandle *FileHandle;

   /* Apertura file: */
   FileHandle = Open("Ray_Trace.pic",(LONG)MODE_NEWFILE);

   if (FileHandle == 0)
   {
      printf("Errore all'apertura file immagine! Numero errore: %ld\n"
                                                           ,IoErr());
      return;
   }

   BitMap = RastPort->BitMap;    /* Puntatore alla strutt. di BitMap */

   /* Determinazione puntatori ai BitPlane: */
   for (i=0; i<BitMap->Depth; i++)
      BitPlanes[i] = (BYTE *)BitMap->Planes[i] - BitMap->BytesPerRow;

   ColorMap = Bildschirm->ViewPort.ColorMap;
   anz_farben = ColorMap->Count;

   /* IFF-Header: */
   Write(FileHandle, "FORM", 4L);        /* "FORM" come introduzione    */
   laenge = 60 + 3*anz_farben +
            BitMap->BytesPerRow * BitMap->Rows * BitMap->Depth;
   Write(FileHandle, &laenge, 4L);       /* Lunghezza del file          */
   Write(FileHandle, "ILBM", 4L);        /* "ILBM" come introduzione    */

   /* Bitmap-Header: */
   Write(FileHandle, "BMHD", 4L);        /* "BMHD" come introduzione    */
   laenge = 20;
   Write(FileHandle, &laenge, 4L);       /* Lunghezza di BMHD           */
   word = BitMap->BytesPerRow * 8;       /*Punti per riga               */
```

```
   Write(FileHandle, &word, 2L);
   Write(FileHandle, &BitMap->Rows, 2L);  /* Punti per colonna         */
   longword = byte = 0;
   Write(FileHandle, &longword, 4L);
   Write(FileHandle, &BitMap->Depth, 1L); /* Profondita' del Plane     */
   Write(FileHandle, &byte, 1L);
   Write(FileHandle, &longword, 4L);
   byte = 10;                            /* Rapporto x-/y              */
   Write(FileHandle, &byte, 1L);
   byte = 11;
   Write(FileHandle, &byte, 1L);
   Write(FileHandle, &word, 2L);
   Write(FileHandle, &BitMap->Rows, 2L);

   /* View-Modes: */
   Write(FileHandle, "CAMG", 4L);         /* "CAMG" come introduzione  */
   longword = 4L;                        /* Lunghezza                  */
   Write(FileHandle, &longword, 4L);
   longword = Bildschirm->ViewPort.Modes;
   Write(FileHandle, &longword, 4L);

   /* Color-Map: */
   Write(FileHandle, "CMAP", 4L);         /* "CMAP" come introduzione  */
   longword = anz_farben * 3;            /* Lungh. definizione colore  */
   Write(FileHandle, &longword, 4L);
   /* Memorizzaz. colori: */
   for (i=0; i<anz_farben; i++)
   {
      farbe = GetRGB4(ColorMap, i);
      byte = ((farbe & 0xf00) >> 4);     /* Percentuale di rosso * 16  */
      Write(FileHandle, &byte, 1L);
      byte = (farbe & 0x0f0);            /* Percentuale di verde * 16  */
      Write(FileHandle, &byte, 1L);
      byte = (farbe & 0x00f) << 4;       /* Percentuale di blu * 16    */
      Write(FileHandle, &byte, 1L);
   }

   /* Dati grafici: */
   Write(FileHandle, "BODY", 4L);         /* "BODY" come introduzione  */
   laenge = BitMap->BytesPerRow * BitMap->Rows * BitMap->Depth;
   Write(FileHandle, &laenge, 4L);

   for (row=0; row<BitMap->Rows; row++)
      for (i=0; i<BitMap->Depth; i++)
         Write(FileHandle, BitPlanes[i] += BitMap->BytesPerRow,
                            (LONG)BitMap->BytesPerRow);

   Close(FileHandle);       /* Chiusura file       */
}


/* Calcolo ampiezza passi per lastra in vetro */
/* e per il vettore sguardo s                 */
VOID get_konst(beo)
struct beobachter *beo;  /* Puntatore alla struttura osservatore */
```

```
{
   FLOAT      zahl;
   FLOAT      l_xe;
   FLOAT      vektor[3];
   REGISTER WORD x_z, z_x, y_z, z_y;

   /* Determinazione vettore sguardo s rispetto alla lastra, */
   /* secondo s = |s|/|Z-B| * (Z-B)                          */
   /* Memorizzazione di s nella struttura dell'osservatore    */
   /*********************************************************/

   /* Determinazione vettore vista allungato.      */
   /* Se gia' indicato, riportare:                 */
   if (beo->blickvek[0] || beo->blickvek[1] || beo->blickvek[2])
   {
      VEKLET(vektor, beo->blickvek);           /* Vettore vista */
   }
   else
   {
      VEKSUB(vektor, beo->blickp, beo->pos);  /* Z-B         */
   }

   zahl = beo->mattsch / VEK LAENGE(vektor);  /* Quoziente   */
   MULVEK(beo->blickv, zahl, vektor);         /* s           */

   /* Calcolo dell'ampiezza passi:  */
   /**********************************/

   /* Calcolo della lunghezza del vettore unitario |xe|: */
   l_xe = 2 * TAN(beo->x_winkel / 2) * beo->mattsch /
            (beo->x_aufl * beo->x_punktgr);

   /* Calcolo di x_step secondo:      */
   /* x_step = xpg * xe (ved. libro) */

   /* Calcolo della coordinata y di x_step:  */
   beo->x_step[1] = 0;     /* poiche' xey = 0 */

   if (beo->blickv[0] != 0  ||
       beo->blickv[2] != 0)          /* sx=0 e sz=0 ? */

   {  /* No: => Casi 1/2: */
      if (beo->blickv[0] != 0)      /* sx<>0 ?        */
      {  /* Si: => Caso 1: */
         x_z = 0;
         z_x = 2;
      }
      else
      {  /* No: => Caso 2: */
         x_z = 2;
         z_x = 0;
      }
```

```
   /* Calcolo delle Coordinate z_x di x_step: */
   /* sz/sx  oppure  sx/sz:                   */
   zahl = beo->blickv[z_x] / beo->blickv[x_z];

   beo->x_step[z_x] = WURZEL(l_xe*l_xe / (1 + zahl*zahl));

   /* Calcolo delle coordinate x_z di x_step: */
   /* -xez*sz/sx  bzw.  -xex*sx/sz          */
   beo->x_step[x_z] = -beo->x_step[z_x] * zahl;
}
else
{  /* Si => Caso 3: */
   beo->x_step[2] = 0;
   beo->x_step[0] = l_xe;
}

/* Moltiplicazione per la dimensione del punto: */
MULVEK(beo->x_step, beo->x_punktgr, beo->x_step);


/* Calcolo di y_step secondo:      */
/* y_step = ypg * ye (Ved. libro) */

/* Calcolo della coordinata x di y_step: */
beo->y_step[0] = 0;     /* wegen yex = 0 */

if (beo->blickv[1] != 0  ||
    beo->blickv[2] != 0)        /* sy=0 e sz=0 ? */

{  /* No: => Casi 1/2: */
   if (beo->blickv[1] != 0)      /* sy<>0 ?         */
   {  /* Si: => Caso 1: */
      y_z = 1;
      z_y = 2;
   }
   else
   {  /* No: => Caso 2: */
      y_z = 2;
      z_y = 1;
   }
   /* Calcolo delle coordinate z_y di y_step: */
   /* sz/sy oppure  sy/sz:                    */
   zahl = beo->blickv[z_y] / beo->blickv[y_z];

   beo->y_step[z_y] = WURZEL(l_xe*l_xe / (1 + zahl*zahl));

   /* Calcolo delle coordinate y_z di y_step: */
   /* -yez*sz/sy  oppure  -yey*sy/sz          */
   beo->y_step[y_z] = -beo->y_step[z_y] * zahl;
}
else
{  /* Si => Caso 3: */
   beo->y_step[2] = 0;
   beo->y_step[0] = l_xe;
}
```

```
   /* moltiplicazione per la grandezza punto: */
   MULVEK(beo->y_step, beo->y_punktgr, beo->y_step);
}


/* Calcolo delle coordinate del */
/* punto attuale sulla lastra   */
VOID get_punkt(beo, x, y, punkt)
struct beobachter *beo;      /* Puntatore struttura osservatore */
LONG     x, y;               /* Coordinate schermo              */
FLOAT    *punkt;

{
   REGISTER WORD xyz;

   /* Calcolo secondo: P = B + s + x*xpg*xe + y*ypg*ye */
   /* dove xpg*xe oppure ypg*ye sono ampiezze passo    */

   for (xyz=0; xyz<3; xyz++)
      punkt[xyz] = beo->blickv[xyz] + beo->pos[xyz] +
                   x*beo->x_step[xyz] + y*beo->y_step[xyz];
}


/***********************************/
/* Routine di Ray-Tracing:         */
/* Determinazione del colore dell'*/
/* oggetto successivo sul raggio   */
/* dello sguardo                   */
/***********************************/

VOID ray_trace(welt, strahl, punkt)
struct welt    *welt;    /* Puntat. struttura mondo */
struct gerade *strahl;  /* Puntat. raggio sguardo  */
struct punkt   *punkt;   /* Puntat. strutt. punto   */

{
   WORD      schnitt_objs();
   VOID      hintergrundfarbe(),
             lichtfarbe(),
             kugelfarbe(),
             flaechenfarbe();

   REGISTER WORD obj_typ; /* Tipo di oggetto incontrato     */
   WORD      *objekt;     /* Puntat. al primo elemento      */
                          /* di una strutt. di oggetto      */

   /* Determinazione dell'Intersezione del raggio */
   /* con l'oggetto successivo                    */
   obj_typ =
   schnitt_objs(welt, punkt, strahl, &objekt);
```

```
/* Determina colore oggetto: */

switch (obj_typ)
{
   case HINTERGRUND:    /* nessun oggetto incontrato */
      hintergrundfarbe(welt, punkt);
      break;
   case LICHTQUELLE:    /* stiamo guardando          */
      lichtfarbe(welt, objekt, punkt);
      break;
   case KUGEL:          /* guardiamo una sfera       */
      kugelfarbe(welt, objekt, strahl, punkt);
      break;
   case FLAECHE:        /* guardiamo una superficie  */
      flaechenfarbe(welt, objekt, strahl, punkt);
      break;
}
}


/* Determinazione del punto di intersezione successivo */
/* di una retta (raggio) con tutti gli oggetti:        */
/* Valore di return: tipo di oggetto incontrato        */
/* in "objekt": Puntatore alla struttura di oggetto:   */
/*              dell'oggetto incontrato                */

WORD schnitt_objs(welt, punkt, strahl, objekt)
struct welt    *welt;   /* Puntat. alla strutt. mondo      */
struct punkt   *punkt;  /* Puntat. alla strutt. punto      */
struct gerade  *strahl; /* Puntat. al raggio               */
WORD     **objekt;      /* Puntat. al Puntat. al primo     */
                        /* elemento di una strutt. oggetto */

{
   BOOL      schnitt_kugel_a(),
             schnitt_flaech();

   REGISTER WORD i;
   WORD      anzahl;
   FLOAT     s_min, s;    /* s piu' piccolo ed s          */
   WORD      *obj;        /* Puntat. al 1 elem. strutt. ogg. */

   s_min = UNENDLICH;                   /* Inizializz. il min. */

   /* Intersezioni con fonti di luce? */
   /***********************************/
   anzahl = welt->anz_licht;           /* Numero fonti luce    */
   for (i=0; i<anzahl; i++)
   {
        /* Calcolo punto intersez. con sfera luminosa */
      if (schnitt_kugel_a(obj=(WORD *)&welt->licht[i], strahl, &s))
      {
         /* L-intersezione esiste: */
         if (s < s_min)
         {
```

```
            s_min = s;                  /* nuovo minimo_        */
            *objekt = obj;              /* annot. strutt. ogg.  */
        }
    }
}

/* Intersezioni con fonti di luce? */
/***********************************/
anzahl = welt->anz_licht;           /* Numero fonti luce    */
for (i=0; i<anzahl; i++)
{
        /* Calcolo punto intersez. con sfera luminosa */
    if (schnitt_kugel_a(obj=(WORD *)&welt->licht[i], strahl, &s))
    {
        /* L-intersezione esiste: */
        if (s < s_min)
        {
            s_min = s;                  /* nuovo minimo         */
            *objekt = obj;              /* annot. strutt. ogg.  */
        }
    }
}

/* Intersezioni con sfere? */
/***************************/
anzahl = welt->anz_kugeln;          /* Numero sfere         */
for (i=0; i<anzahl; i++)
{
        /* Calcolo intersezione con sfera */
    if (schnitt_kugel_a(obj=(WORD *)&welt->kugeln[i], strahl, &s))
    {
        /* Intersezione esiste: */
        if (s < s_min)
        {
            s_min = s;                  /* nuovo minimo         */
            *objekt = obj;              /* annot. strutt. ogg.  */
        }
    }
}

/* Intersezioni con superfici? */
/*******************************/
anzahl = welt->anz_flaech;          /* Numero superfici     */
for (i=0; i<anzahl; i++)
{
        /* Calcolo intersezioni con superf. */
    if (schnitt_flaech(obj=(WORD *)&welt->flaech[i], strahl, &s))
    {
        /* Intersezione esiste: */
        if (s < s_min)
        {
            s_min = s;                  /* nuovo minimo         */
            *objekt = obj;              /* Annot. strutt. ogg.  */
```

```
			VEKLET(punkt->pos, sch_p); /* Annot. intersez.   */
			punkt->user1 = sch_m;     /* annot. di m e k    */
			punkt->user2 = sch_k;
		}
	}
}

/* Nessuna intersezione? */
if (s_min == UNENDLICH)
	/* quindi sfondo: */
	*objekt = (WORD *)&welt->hinten[0];

/* Trasferimento Tipi oggetti ed s_min: */
punkt->s = s_min;
return (**objekt);
}


/* Calcolo punto di intersezione: */
/************************************/

/* Sfera:                                        */
/* Determinazione dell'intersezione di una sfera */
/* con un raggio che arrivi sulla sfera          */
/* dall'esterno (Retta)                          */
/* Risposta: l'intersezione esiste/non esiste    */
/*         oltre la lastra di vetro              */

BOOL schnitt_kugel_a(kugel, strahl, s)
struct kugel    *kugel;      /* Puntatore alla strutt. sfera */
struct gerade   *strahl;     /* Puntatore alla retta         */
FLOAT      *s;               /* Puntatore alla s ricercata   */

{
	FLOAT      v[3], *Pm, *bv, *Km;
	FLOAT      a, b, c, D;

	Pm = &strahl->p0[0];        /* Punto PM e' P0 della retta   */
	Km = &kugel->pos[0];        /* Punto Km e' il centro sfera  */
	bv = &strahl->a[0];         /* bv e' vettore direz. retta   */

	VEKSUB(v, Pm, Km);          /* v = Pm-Km                    */

	/* Calcolo dei coefficienti per la   */
	/* equazione quadrata di s:          */
	a = SKAPRD(bv, bv);         /* a = bx^2 + by^2 + bz^2       */
	b = 2*SKAPRD(v, bv);        /* b = 2*(vxbx + vyby + vzbz)   */
	c = SKAPRD(v, v) -          /* c = vx^2 + vy^2 + vz^2 - R^2 */
		kugel->radius * kugel->radius;

	/* Calcolo della discriminante per s: */
	D = b*b - 4*a*c;            /* D = b^2 - 4*a*c              */
```

```
    /* Esiste una intersezione? */
    if (D>0)
    {
       /* si: */
       D = WURZEL(D);
       *s = (-b-D)/(2*a);     /* calcolo dell's minore         */

       if (*s <= MINI_S)      /* s troppo piccolo ?            */
          *s = (-b+D)/(2*a);  /* si=> calcolare un s maggiore */

       return (BOOL)(*s>MINI_S); /* s ancora troppo piccolo? */
    }
    else
    {
       return (BOOL)(FALSE); /* Nessuna intersez. presente    */
    }
}


/* Superficie:                                         */
/* Calcolo del punto di intersezione di una            */
/* retta (raggio) con un parallelogramma               */
/* (esempio: rettangolo o quadrato)                    */
/* si ricerca s come fattore di allungamento           */
/* per il vettore di direzione della retta             */

BOOL schnitt_flaech(flaeche, strahl, s)
struct flaeche *flaeche;     /* Puntatore strutt. superficie */
struct gerade  *strahl;      /* Puntatore strutt. retta      */
FLOAT    *s;                 /* Puntatore alla s ricercata   */

{
    REGISTER WORD i, j;

    FLOAT     *n;            /* Vettore normale superficie   */
    FLOAT     *P0;           /* Punto fisso sulla superficie */
    FLOAT     *v, *w;        /* Vettori di direzione superf. */
    FLOAT     *Pm, *b;       /* Parametro della retta        */
    FLOAT     vektor[3];     /* Parametro intermedio         */
    FLOAT     nenner, q;

    P0 = &flaeche->ebene.p0[0];
    v  = &flaeche->ebene.a[0];
    w  = &flaeche->ebene.b[0];

    Pm = &strahl->p0[0];
    b  = &strahl->a[0];

    n = &flaeche->ebene.n[0];

    /* In caso di necessita', calcolo vettore normale n: */
    if (NOT flaeche->ebene.n_valid)
    {
       VEKPRD(n, v, w);                /* n = v x w        */
       flaeche->ebene.n_valid = TRUE;
    }
```

```
  /* Se il nominatore (b*n) e' diverso da zero      */
  /*          => l'intersez. col piano esiste       */
  /* Se il nominatore e' uguale a zero              */
  /*          => nessuna intersez. col piano        */

  if ( (nenner = SKAPRD(b, n)) != 0.0)
  {
     /* Determinaz. intersez. con s:                */
     VEKSUB(vektor, P0, Pm);
     *s = SKAPRD(vektor, n) / nenner;

     /* Calcolo delle coordinate del punto:         */
     GERADEN_PKT(sch_p, Pm, *s, b);


     /* Il punto si trova nel parallelogramma?      */

     /* Ricerca coppia di equazioni adeguata        */
     for (i=2; i>=0; i--)
        for (j=0; j<3; j++)

           /* vi<>0 und wj*vi-wi*vj<>0 ? */
           if ( i!=j && v[i] && (nenner=w[j]*v[i]-w[i]*v[j]) )
              goto weiter;

     return (FALSE);           /* se errore, indietro!      */

     weiter:  /* i e j sono i numeri delle equazioni        */

     /* Calcolo dei fattori del piano m e k: */
     sch_m = ( (sch_p[j]-P0[j]) * v[i] - (q=sch_p[i]-P0[i]) * v[j] )
             / nenner;
     if (sch_m >= 0.0  &&  sch_m <= 1.0)
        sch_k = (q - sch_m*w[i]) / v[i];
     else
        return (FALSE);          /* Intersezione all'esterno  */
                                 /* del parallelogramma       */
     if (sch_k >= 0.0  &&  sch_k <= 1.0)
        return (BOOL)(*s>MINI_S); /* s troppo piccolo?        */
     else
        return (BOOL)(FALSE);  /* Intersezione all'esterno   */
                               /* del parallelogramma        */
  }
  else
  {
     return (BOOL)(FALSE);     /* nessuna intersezione       */
  }
}


/* Calcolo colori */
/******************/

/* Sfondo: */
```

```
VOID hintergrundfarbe(welt, punkt)
struct welt     *welt;   /* Puntatore alla struttuta mondo     */
struct punkt    *punkt;  /* Puntatore alla struttura punto     */

{
   struct hintergrund *hg; /* Puntatore alla strutt. sfondo */

   hg = &welt->hinten[0];

   /* Prelevam. colore sfondo: */
   COLLET(punkt->farbe, hg->farbe1);
}


/* Fonte di luce: */

VOID lichtfarbe(welt, lq, punkt)
struct welt        *welt;   /* Puntat. strutt. mondo   */
struct lichtquelle *lq;  /* Puntat. strutt. luce    */
struct punkt       *punkt;  /* Puntat. strutt. punto   */

{
   /* Prelevam. colore luce: */
   COLLET(punkt->farbe, lq->farbe);
}


/* Sfera: */

VOID kugelfarbe(welt, kugel, strahl, punkt)
struct welt     *welt;    /* Puntat. strutt. mondo   */
struct kugel    *kugel;   /* Puntat. strutt. sfera   */
struct gerade   *strahl;  /* Puntat. strutt. raggio */
struct punkt    *punkt;   /* Puntat. strutt. punto   */

{
   VOID      farbintens();

   FLOAT     normale[3]; /* Vettore normale sfera   */

   /* Calcolo ulteriore intersezione: */
   GERADEN_PKT(punkt->pos, strahl->p0, punkt->s, strahl->a);

   /* Calcolo vettore normale: */
   /* secondo: n = Intersezione  - centro sfera */
   VEKSUB(normale, punkt->pos, kugel->pos);

   /* Determinazione del colore di base del punto (Koh/Kod): */
   COLLET(punkt->farbe, kugel->material.farbe);

   /* Determinazione intensita' colore: */
   farbintens(welt, normale, strahl, punkt, &kugel->material);

}
```

```
/* Superficie: */

VOID flaechenfarbe(welt, flaeche, strahl, punkt)
struct welt    *welt;    /* Puntat. strutt. mondo   */
struct flaeche *flaeche; /* Puntat. strutt. superf. */
struct gerade  *strahl;  /* Puntat. strutt. raggio  */
struct punkt   *punkt;   /* Puntat. strutt. punto   */

{
   VOID      farbintens();

   REGISTER UWORD zahl1, zahl2;
   FLOAT          fzahl1, fzahl2;
   struct material *mat;

   mat = &flaeche->material;

   /* Determinaz. colore di base del punto (Koh/Kod):       */
   switch (mat->oberf_typ) /* a seconda del tipo di superf. */
   {
      case 0: /* normale, tinta unita:  */
         COLLET(punkt->farbe, mat->farbe);
         break;

      case 1: /* a quadri, bicolore:    */
         /* Determinazione del campo:   */
         /* INT(m * anzahl_karos_m) modulo 2) fornisce */
         /* il tipo di campo (0 o 1) in direzione m    */
         /* INT(k * anzahl_karos_k) moduló 2) fornisce */
         /* il tipo di campo (0 o 1) in direzione k    */

         zahl1 =
         ((ULONG)(punkt->user1 * mat->user.muster.anz_m
                               + 0.001)) % 2; /* m */
         zahl2 =
         ((ULONG)(punkt->user2 * mat->user.muster.anz_k
                               + 0.001)) % 2; /* k */
         if ( (zahl1 + zahl2) % 2 )
         {
            COLLET(punkt->farbe, mat->farbe);
         }
         else
         {
            COLLET(punkt->farbe, mat->farbe2);
         }
         break;

      case 2: /* Definizione di 16x16 motivi                   */
         /* Determinazione della striscia attuale di bit      */
         /* zahl = k*anz_muster_k oppure zahl=m*anz_muster_m */
         /* numero strisce = INT(16* (zahl-INT(zahl) ))       */

         /* m: */
         zahl1 =
```

```
        fzahl1 = punkt->user1 * mat->user.muster.anz_m + 0.001;
        zahl1 =
        fzahl1 = 16 * (fzahl1 - (FLOAT)zahl1 );

        /* k: */
        zahl2 =
        fzahl2 = punkt->user2 * mat->user.muster.anz_k + 0.001;
        zahl2 =
        fzahl2 = 16 * (fzahl2 - (FLOAT)zahl2 );

        /* a seconda dell-impostazione o no del bit          */
        /* applicare nel motivo del punto il colore 1 o 2   */
        if ( mat->user.muster.muster[zahl1] &  (1L << zahl2) )
        {
           COLLET(punkt->farbe, mat->farbe);
        }
        else
        {
           COLLET(punkt->farbe, mat->farbe2);
        }
        break;
   }

   /* Determinazione intensita' colore: */
   farbintens(welt, flaeche->ebene.n, strahl, punkt, mat);

}


/* Legge di riflessione: Calcolo del vettore di riflessione */
/* dal vettore di incidenza e dalla normale alla superficie */

VOID reflexion(refl_vek, einf_vek, normale)
FLOAT     *refl_vek;          /* Puntat. al vettore di rifless.   */
FLOAT     *einf_vek;          /* Puntat. al vettore di incidenza  L */
FLOAT     *normale;           /* Puntat. al vettore della normale */

{
   FLOAT     Lp[3];           /* Vettore incidenza proiettato     */
   FLOAT     q;

   /* 2 * vettore incidenza proiettato: Lp=2*(L*n)/n^2 * n */
   q = 2 * SKAPRD(einf_vek, normale) / SKAPRD(normale, normale);
   MULVEK(Lp, q, normale);

   /* Determina il vettore di riflessione secondo: R=Lp-L */
   VEKSUB(refl_vek, Lp, einf_vek);
}

/**************************************************************/
/* Calcolo intensita' di colore di un punto di un oggetto */
/**************************************************************/
```

```
VOID farbintens(welt, normale, strahl, punkt, material)
struct welt      *welt;     /* Puntat. struttura mondo          */
FLOAT            *normale;  /* Puntat. normale superficie       */
struct gerade    *strahl;   /* Puntat. strutt. raggio           */
struct punkt     *punkt;    /* Puntat. strutt. punto            */
struct material *material;  /* Puntat. Strutt. materiale        */

{
   VOID      ray_trace(),
             reflexion();
   WORD      schnitt_objs();
   FLOAT     potenz();

   struct gerade  refl_strahl;  /* Raggio riflessione  */
   struct punkt   refl_punkt;   /* Punto riflessione   */
   FLOAT     farbe[3];

   /* Inizializzazione valore colore */
   farbe[0] = farbe[1] = farbe[2] = 0.0;


   /* Elaborazione specchiature: */
   /********************************/

   if (material->licht_typ == VERSPIEGELT)
   {
      /* Oggetto specchiato                */

      /* Intensita' luce specchiata        */
      {
         REGISTER WORD  rgb;

         for (rgb=0; rgb<3; rgb++)
         {
            refl_punkt.intens_b[rgb] =
            punkt->intens_b[rgb] * material->spiegel_int[rgb];
         }
      }

/* Criterio di interruzione: la luce specchiata e'   */
/* scura oppure: e' stato raggiunto il numero max.   */
/* di recursioni                                      */
if ( (refl_punkt.intens_b[0] > SCHWARZ ||
      refl_punkt.intens_b[1] > SCHWARZ ||
      refl_punkt.intens_b[2] > SCHWARZ    ) &&
      rekur_zaehl < REKUR_MAX )
{
   /* La luce e' ancora sufficientemente chiara        */
   /* Recursione! Il raggio di riflessione viene       */
   /* ancora seguito. Si cerca il colore!              */

   /* Calcola raggio di riflessione: */
   refl_strahl.a[0] = -strahl->a[0]; /* Raggio incidente */
   refl_strahl.a[1] = -strahl->a[1]; /* da ruotare       */
```

```
      refl_strahl.a[2] = -strahl->a[2];
      reflexion(refl_strahl.a, refl_strahl.a, normale);

      /* Punto fisso del nuovo raggio = intersezione: */
      VEKLET(refl_strahl.p0, punkt->pos);

      /* Incremento contatore recursioni: */
      rekur_zaehl++;

      /* Eccola: */
      ray_trace(welt, &refl_strahl, &refl_punkt);

      /* Decremento contatore recursioni: */
      rekur_zaehl--;

      /* Trasferimento colore determinato               */
      /* tenendo conto dell'intensita' di specchiatura: */
      {
         REGISTER WORD rgb;

         for (rgb=0; rgb<3; rgb++)
            farbe[rgb] = refl_punkt.farbe[rgb] *
                             material->spiegel_int[rgb];
      }
   }
}


/* Riflessione della luce diffusa e a specchio: */
/************************************************/

{ /* Nuovo inizio blocco con alcune variabili locali:      */
   REGISTER WORD rgb, lq;
   struct gerade licht_strahl; /* Raggio verso la luce    */
   struct lichtquelle *licht;  /* Puntat.str. fonte luce */
   FLOAT     cos_a, cos_b;
   FLOAT     nenner, potenz;
   WORD      *z_objekt;

   /* Punto fisso dei raggi della luce e riflessi: */
   VEKLET(licht_strahl.p0, punkt->pos);
   VEKLET(refl_strahl.p0, punkt->pos);

   /* Elaborazione di tutte le fonti di luce: */
   for (lq=0; lq < welt->anz_licht; lq++)
   {
      /* Indirizzo strutt. attuale fonte luce  */
      licht = &welt->licht[lq];

      /* Determina intersezione raggio->Fonte di luce:     */
      /* P0=Intersezione, a=(fonte luce - intersezione)    */
      VEKSUB(licht_strahl.a, licht->pos, punkt->pos);

      /* Esiste intersezione davanti alla fonte di luce? */
      schnitt_objs(welt, &refl_punkt, &licht_strahl, &z_objekt);
```

```
if (z_objekt == (WORD *)licht)
{
    /* no: l'oggetto e' illuminato dalla fonte luce */
    /*        => calcolare l'intensita' luminosa:     */

    /* Determina il vettore di riflessione luce R:  */
    reflexion(refl_strahl.a, licht_strahl.a, normale);


    /* Calcola i valori dei colori in RGB del punto */
    /* per la fonte di luce:                        */

    /* Formula della intensita'                     */
    /* intens =                                     */
    /* Iq * [Kod*cos(a) + Kr*cos(b)^f] / (d * Kd)   */
    /* dove:                                        */
    /*      Iq      = licht->farbe[]                */
    /*      Kod     = punkt->farbe[]                */
    /*      cos(a)  = n'*L' = n*L/(|n|*|L|)         */
    /*      n       = normale                       */
    /*      L       = licht_strahl.a                */
    /*      Kr      = material->spieg_ref[]         */
    /*      cos(b)  = R'*S' = R*S/(|R|*|S|)         */
    /*      R       = refl_strahl.a                 */
    /*      S       = - strahl->a                   */
    /*      f       = material->fokus_ref[]         */
    /*      d       = VEK_LAENGE(licht_strahl.a)    */
    /*      Kd      = licht->dist_konst             */

    nenner = VEK_LAENGE(licht_strahl.a) *
             licht->dist_konst;

    /* Calcolo del Coseno: */
    cos_a = SKAPRD(normale, normale) *
            SKAPRD(licht_strahl.a, licht_strahl.a);
    cos_a = SKAPRD(normale, licht_strahl.a) /
            WURZEL(cos_a);

    cos_b = SKAPRD(refl_strahl.a, refl_strahl.a) *
            SKAPRD(-strahl->a, -strahl->a);
    cos_b = SKAPRD(refl_strahl.a, -strahl->a) /
            WURZEL(cos_b);

    for (rgb=0; rgb<3; rgb++)
    {
       potenz = POTENZ(cos_b, material->fokus_ref[rgb]);
       farbe[rgb] += licht->farbe[rgb] *
         ( punkt->farbe[rgb] * cos_a +
           material->spieg_ref[rgb] * potenz
         ) / nenner;

    } /* for rgb */
  } /* if */
} /* for lq */
```

```
   /* Presa in considerazione della luminosita' di         */
   /* sottofondo secondo: I = Ih*Koh + altre intensita'    */
   for (rgb=0; rgb<3; rgb++)
   {
      punkt->farbe[rgb] =
      farbe[rgb] + welt->hintlicht[rgb] * punkt->farbe[rgb];
      }
   } /* Fine blocco */
}

/* Tavolozza colori inizializzata: */
VOID init_farben()

{
   REGISTER UWORD reg, rgb;

   for (reg=0; reg<ANZ_PAL_FAB; reg++)
   {
      /* Impostazione tavolozza colori: */
      SetRGB4(&Bildschirm->ViewPort, (LONG)reg,
             palette[reg][0],
             palette[reg][1],
             palette[reg][2] );

      /* Trasformazione di valori Hardware di Intensita'     */
      /* in valori interni al programma (da 0 - 2^16):       */
      for (rgb=0; rgb<3; rgb++)
      {
         if (G_MOD & EXTRA_HALFBRITE)
         {
            /* i registri colore superiori hanno mezza intensita': */
            /* calcolare l'intero della mezza intensita':          */
            palette[reg+32][rgb] =
              ((palette[reg][rgb] & 0xfffffffeL) << 15 ) / ANZ_INT;
         }

         palette[reg][rgb] = (palette[reg][rgb] << 16) / ANZ_INT;
      }
   }
}

/* Traccia un punto sullo schermo:          */
/* l'intensita' RGB viene trasferita,       */
/* viene ricercata la tavolozza colori      */
/* piu' vicina al valore RGB                */

VOID plot(x, y, punkt)
LONG           x, y;    /* Coordinate schermo del punto        */
struct punkt   *punkt;  /* Puntatore struttura di punto        */

{
   WORD     zufall();
```

```
REGISTER UWORD reg, rgb;
UWORD    reg_min=0, zweit_reg_min=0;

ULONG    farbe[3];

ULONG    diff_sum_min = 0x7fffffff,  /* valore max.poss. */
         zweit_sum_min = 0x7fffffff,
         diff1_sum,
         diff2_sum;
ULONG    diff max min = 0x7fffffff,
         zweit_max_min = 0x7fffffff,
         diff1_max,
         diff2_max;

UWORD    reg2;
LONG     diff;

/* Fissazione colore punto fra 1 e 2          */
/* e trasformazione in formato tavolozza      */
for (rgb=0; rgb<3; rgb++)
{
   punkt->farbe[rgb] =
          MAX(0.0, punkt->farbe[rgb]);
   punkt->farbe[rgb] =
          MIN(1.0, punkt->farbe[rgb]);
   farbe[rgb] = (ULONG)(punkt->farbe[rgb] * 65536);
}


/* Ricerca del registro di tavolozza adeguato  */

/* per EXTRA_HALFBRITE: doppio gruppo colori     */
for (reg=0; reg<ANZ_FARBEN; reg++)
{
   diff1_sum = 0;
   diff2_sum = 0;
   diff1_max = 0;
   diff2_max = 0;

   for (rgb=0; rgb<3; rgb++)
   {
      /* Somma delle differenze delle intensita' RGB: */

      diff = palette[reg][rgb] - farbe[rgb];
      diff1_sum += diff = ABS(diff);

      /* e differenza massima: */
      diff1_max = MAX(diff1_max, diff);
   }

   diff2_sum = diff1_sum;
   diff2_max = diff1_max;
   reg2      = reg;
```

```
      /* Calcolo del minimo: */
      if (diff_sum_min > diff1_sum  ||
         (diff_sum_min == diff1_sum  &&  diff_max_min > diff1_max) )
      {
         diff2_sum = diff_sum_min;
         diff2_max = diff_max_min;
         reg2      = reg_min;
         diff_sum_min = diff1_sum;   /* Nuovo minimo          */
         diff_max_min = diff1_max;
         reg_min      = reg;         /* Annotare il registro */
      }

      /* Calcolo del penultimo minimo: */
      if ( (zweit_sum_min > diff2_sum  ||
            (zweit_sum_min == diff2_sum  &&  zweit_max_min > diff2_max) )
            && diff2_sum > diff_sum_min )
      {
         zweit_sum_min = diff2_sum; /* Nuovo minimo          */
         zweit_max_min = diff2_max;
         zweit_reg_min = reg2;      /* Annotare il registro */
      }
   }

   /* Selezionare la piu' piccola somma e la penultima       */
   /* casualmente con verosimiglianza ponderata come fonte  */
   /* per il registro di colori valido di volta in volta    */
   if ( (FLOAT)zufall() + (zweit_sum_min << 8) /
                 (zweit_sum_min+diff_sum_min) * 1.20 > 0xff )
   {
      /* Impostazione colore: */
      SetAPen(RastPort, (LONG)reg_min);
   }
   else
   {
      /* Impostazione colore: */
      SetAPen(RastPort, (LONG)zweit_reg_min);
   }

   /* Tracciatura dei punti: */
   /*****************************/
     WritePixel(RastPort, x, y);
   /*****************************/
}


/* Determinazione di un numero intero casuale */
/* fra 0 e $100                               */

WORD zufall()
{
   static LONG zuf = 100001;

   zuf *= 125;
   zuf %= 2796203;
   return (WORD)( (zuf << 8)/2796203);
}
```

```
#include <exec/types.h>
#include <intuition/intuition.h>
#include <libraries/mathffp.h>

#define ANZ_FARBEN 32

/* Dichiarazioni di funzioni (solo per Compilatore Aztec): */
/* a scelta anche : #include <functions.h>                 */
/*********************************************************/
VOID                    ClearScreen();
VOID                    Exit();
struct Message *        GetMsg();
VOID                    Move();
struct Library *        OpenLibrary();
struct Screen *         OpenScreen();
struct Window *         OpenWindow();
VOID                    ReplyMsg();
VOID                    SetAPen();
VOID                    SetRGB4();
LONG                    Wait();
VOID                    WritePixel();
/*****************************************************/


/* Colori da tavolozza: */
ULONG palette[ANZ_FARBEN][3] =
   {
      { 1, 1, 1}, {15,13,13},
      {15,15,15}, {15,10,10},
      {13,13,13}, {15, 4, 4},
      {11,11,11}, {15, 0, 0},
      { 9, 9, 9}, {14, 0, 0},
      { 0, 0, 4}, {13, 0, 0},
      { 7, 7, 7}, {11, 0, 0},
      { 5, 5, 5}, { 9, 0, 0},
      { 4, 4, 4}, { 7, 0, 0},
      { 3, 3, 3}, { 5, 0, 0},
      { 2, 2, 2}, { 4, 0, 0},
      {13,13,15}, { 0, 0,13},
      {10,10,15}, { 0, 0,11},
      { 6, 6,15}, { 0, 0, 9},
      { 0, 0,15}, { 0, 0, 7},
      { 0, 0,14}, { 0, 0, 5},
   }; /* inizializzare solo i primi 32 Colori */


#define WURZEL SPSqrt
#define SIN    SPSin
#define COS    SPCos
#define TAN    SPTan
#define LOG    SPLog
#define EXP    SPExp
```

```
/******************/
/* Hauptprogramm */
/******************/

/* Definizioni e macro: */

/* Determinazione del massimo/minimo di due valori: */
#define MAX(a,b)   ( (a) > (b) ? (a) : (b) )
#define MIN(a,b)   ( (a) < (b) ? (a) : (b) )

/* Determinazione del valore assoluto di un numero: */
#define ABS(a) ( (a) < 0  ?  -(a) : (a) )

/* Determinazione della potenza a^b:     */
/* nach p=a^b <=> p=e^(b*ln(a)) */
#define POTENZ(a,b) (EXP( (b)*LOG(a) ))


/* Operazioni vettoriali come macro:  */
/* a, b, c devono essere nomi vettori */

/* Attribuzione di un vettore  a=b: */
#define VEKLET(a,b) (a[0]=(b[0]); a[1]=(b[1]); a[2]=(b[2]);)

/* Attribuzione di una matrice colori a=b */
#define COLLET  VEKLET

/* Somma vettoriale c=a+b di due vettori */
#define VEKADD(c,a,b) (c[0] = (a[0]) + (b[0]);\
                       c[1] = (a[1]) + (b[1]);\
                       c[2] = (a[2]) + (b[2]);)

/* Sottrazione vettoriale c=a-b di due vettori */
#define VEKSUB(c,a,b) (c[0] = (a[0]) - (b[0]);\
                       c[1] = (a[1]) - (b[1]);\
                       c[2] = (a[2]) - (b[2]);)

/* Moltiplicazione b=z*a di un vettore per un numero */
#define MULVEK(b,z,a) (b[0] = (z) * (a[0]);\
                       b[1] = (z) * (a[1]);\
                       b[2] = (z) * (a[2]);)

/* Prodotto scalare a*b di due vettori */
#define SKAPRD(a,b) ( (a[0])*(b[0]) +\
                      (a[1])*(b[1]) +\
                      (a[2])*(b[2]) )

/* Prodotto vettoriale c=axb di due vettori */
#define VEKPRD(c,a,b) (c[0] = (a[1])*(b[2]) - (a[2])*(b[1]);\
                       c[1] = (a[2])*(b[0]) - (a[0])*(b[2]);\
                       c[2] = (a[0])*(b[1]) - (a[1])*(b[0]);)

/* Determinazione dell'equazione di una retta P = P0 + s*a */
/* nella struttura struct gerade g (strutt retta)          */
```

```
/* da due punti b e c: P0=b, a=b-c                              */
/* Con cio' la vetta del vettore di direzione                   */
/* e' contemporaneamente punto fisso della retta                */

#define GET_GERADE(g,b,c) (VEKLET(g.p0,b); VEKSUB(g.a,b,c);)


/* Calcolo di un punto dalla equazione */
/* vettoriale di una retta             */
/* P = P0 + s*a                        */

#define GERADEN_PKT(P,P0,s,a) (P[0] = (P0[0]) + (s)*(a[0]);\
                               P[1] = (P0[1]) + (s)*(a[1]);\
                               P[2] = (P0[2]) + (s)*(a[2]);)

/* Calcolo della lunghezza di un vettore a: */
#define VEK_LAENGE(a)  (WURZEL(SKAPRD(a,a)))


/* Caratteristiche oggetto (Tipi): */
#define HINTERGRUND  0
#define LICHTQUELLE  1
#define KUGEL        2
#define FLAECHE      3

/* Operazioni per la luce: */
#define NORMAL       0
#define VERSPIEGELT  1
#define DURCHSICHTIG 2

/* altri parametri: */
#define UNENDLICH     1.0e10     /* Lontano                   */
#define SCHWARZ       0.05       /* Limiti luce               */
#define MINI_S        0.0001     /* s minimo permesso         */
                                 /* (quasi 0 a causa della    */
                                 /*  imprecisione di calcolo) */


/* Variabili e strutture globali: */
/**********************************/

/* Singolo punto dell'oggetto: */
struct punkt
{
   FLOAT     pos[3];       /* Posizione del punto      */
   FLOAT     s;            /* Valore s per eq. retta */
   FLOAT     farbe[3];     /* Colore                   */
   FLOAT     intens_b[3];/* Intensita' RGB             */
   FLOAT     user0[3];     /* altri valori             */
   FLOAT     user1;
   FLOAT     user2;
};
```

```
/* Parametro retta secondo P = P0 + s*a: */
struct gerade
{
   FLOAT     p0[3];       /* Punto fisso P0          */
   FLOAT     a[3];        /* Vettore direzione a     */
};

/* Parametro piano secondo P = P0 + s*a + t*b: */
struct ebene
{
   FLOAT     p0[3];       /* Punto fisso P0          */
   FLOAT     a[3];        /* Vettore direzione a     */
   FLOAT     b[3];        /* Vettore direzione b     */
   BOOL      n_valid;     /* Flag: TRUE => Vettore   */
                          /*     normale n presente */
                          /*       FALSE => n non    */
                          /*        e' presente      */
   FLOAT     n[3];        /* Vettore normale piano   */
};

/* Costanti materiale per un oggetto: */
struct material
{
   FLOAT     farbe[3];          /* Colore RGB di un oggetto         */
   FLOAT     farbe2[3];         /* eventuale secondo colore         */
   FLOAT     farbe3[3];         /* eventuale terzo colore           */
                                /* assumono il ruolo di             */
                                /* costanti Koh e Kod               */
   FLOAT     spieg_ref[3];      /* Costanti per la riflessione      */
                                /* a specchio Kr                    */
   FLOAT     fokus_ref[3];      /* Messa a fuoco della rifles-      */
                                /* sione a specchio f               */
   FLOAT     spiegel_int[3];/* Intensita' della luce            */
                                /* specchiata                       */
   BYTE      licht_typ;         /* Tipo operazione luce             */
                                /* =0: Normale                      */
                                /* =1: Totalmente specchiata        */
                                /* =2: trasparente                  */
   BYTE      oberf_typ;         /* Tipo superficie                  */
                                /* =0: Normale, tinta unita         */
                                /* =1: esempio a riquadri           */
   union                        /* Parametri individuali           */
   {
      struct
      {
         WORD anz_k,anz_m;  /* Numero motivi in direz. k/m      */
         WORD muster[16];   /* Definizione motivo               */
      } muster;
      struct
      {                         /* oppure intero                    */
         WORD user1[3];
         WORD user2[3];
         WORD user3[3];
```

```
      } int_user;
     .struct
      {                           /* oppure di tipo Float             */
         FLOAT user1[3];
         FLOAT user2[3];
         FLOAT user3[3];
      } float_user;
   } user;
};

/* Sfondo all'"infinito" */
struct hintergrund
{
   WORD      typ;         /* Tipo per sfondo: 0      */
   FLOAT     unendlich;   /* Valore per posizione    */
   FLOAT     farbe1[3];   /* Colore 1                */
   FLOAT     farbe2[3];   /* Colore 2                */
};

/* Fonte di luce: */
struct lichtquelle
{
   WORD      typ;         /* Tipo per fonte luce: 1 */
   FLOAT     pos[3];      /* Posiz. fonte di luce   */
   FLOAT     radius;      /* Raggio                 */
   FLOAT     farbe[3];    /* Colore della luce      */
   FLOAT     dist_konst;  /* Costante distanza Kd   */
};

/* Sfera: */
struct kugel
{
   WORD      typ;         /* Tipo per sfera: 2      */
   FLOAT     pos[3];      /* Posizione centro       */
   FLOAT     radius;      /* Raggio                 */
   struct material material; /* Costanti mater.    */
};

/* Superficie (Parallelogramma): */
struct flaeche
{
   WORD      typ;          /* Tipo per superf.: 3   */
   FLOAT     p1[3],p2[3],  /* tre punti nagolari    */
             p3[3];
   struct ebene ebene;     /* Parametri piani con   */
                           /* vettore della normale */
   struct material material; /* Costanti mater.    */
};

/* Osservatore: */
struct beobachter
{
   FLOAT     pos[3];      /* Posizione osservatore            */
```

```
   FLOAT     blickp[3];  /* Punto al quale sta guardando      */
   FLOAT     blickvek[3];/* In alternativa: vettore vista     */
                         /* se il vettore vista e' = 0,       */
                         /* verra' utilizzato il punto        */
                         /* cui si sta guardando              */
   FLOAT     mattsch;    /* Distanza rispetto lastra vetro    */
   FLOAT     x_punktgr;  /* Larghezza punto                   */
   FLOAT     y_punktgr;  /* Altezza punto                     */
   LONG      x_aufl;     /* Risoluzione x                     */
   LONG      y_aufl;     /* Risoluzione y                     */
   FLOAT     x_winkel;   /* Angolo vista orizzontale          */
   /* variabili da calcolare in seguito: */
   FLOAT     blickv[3];  /* Vettore sguardo con lunghezza     */
                         /* Osservatore -> lastra vetro       */
   FLOAT     x_step[3];  /* Vettore ampiezza passo x          */
   FLOAT     y_step[3];  /* Vettore ampiezza passo y          */
};

/* Tutto il mondo: */
struct welt
{
   WORD      anz_obj;           /* Numero di tutti gli ogg.   */
   WORD      anz_kugeln;        /* Numero di tutte le sfere   */
   struct kugel *kugeln;        /* Tutte le sfere             */
   WORD      anz_flaech;        /* Numero di tutte le superf. */
   struct flaeche *flaech;      /* Tutte le superfici         */
   WORD      anz_licht;         /* Numero di tutte le luci    */
   struct lichtquelle *licht;   /* Tutte le fonti di luce     */
   WORD      anz_hinten;        /* Numero degli sfondi = 1    */
   struct hintergrund *hinten;/* ampiezza, il Nulla           */
   FLOAT     hintlicht[3];      /* Illuminaz. di sottof. Ih   */
};


/******************************/
/* Dati globali per il mondo */
/******************************/

/* Mondo circostante: */
struct new_welt
{
   FLOAT     hintlicht[3];
} new_welt = { {0.32, 0.32, 0.32} };

/* Osservatore: */
struct beobachter new_beo =
{
   {21.0, 24.0,-29.0},        /* Posizione                     */
   { 0.0,  0.0,  0.0},        /* Punto cui sta guardando       */
   {-1.5, -1.5,  3.0},        /* alternativa: vettore vista    */
   1.0,                       /* Distan. rispetto lastra vetro */
   1.00, 1.00,                /* Larghezza-altezza punto       */
   319L, 245L,                /* Risoluzione x-/y              */
   55.0,                      /* Angolo visuale orizz. (Gradi) */
```

```
  {0.0, 0.0, 0.0},          /* ai valori inizializzati          */
  {0.0, 0.0, 0.0},
  {0.0, 0.0, 0.0}
};

/* Sfondo: */
struct hintergrund new_hint =
{
  HINTERGRUND,              /* Tipo         */
  UNENDLICH,                /* "Posizione"  */
  {0.00, 0.00, 0.80},       /* Colore 1     */
  {0.00, 0.00, 0.00}        /* Colore 2     */
};

/* Fonti di luce: */
struct lichtquelle new_licht[] =
{
   {
    LICHTQUELLE,              /* Tipo         */
    {23.5,20.0, 20.0},        /* Posizione    */
     1.0,                     /* Raggio       */
    {1.00, 1.00, 1.00},       /* Colore       */
     0.18                     /* Kd           */
   },
   {
    LICHTQUELLE,              /* Tipo         */
    { 7.0, 1.0, -6.0},        /* Posizione    */
     1.0,                     /* Raggio       */
    {1.00, 1.00, 1.00},       /* Colore       */
     1.9                      /* Kd           */
   }
};

/* Oggetti: */

/* 1. Sfere: */
struct kugel new_kugeln[] =
{
   {
    KUGEL,                    /* Tipo         */
    { 6.0,  6.5,  3.0},       /* Posizione    */
    3.0,                      /* Raggio       */
    { /* Material: */
     {0.00, 0.00, 0.00},      /* Colore       */
     {0.00, 0.00, 0.00},      /* Colore 2     */
     {0.00, 0.00, 0.00},      /* Colore 3     */
     {0.00, 0.00, 0.00},      /* Kr           */
     { 5.0,  5.0,  5.0},      /* Fuoco f      */
     {0.85, 0.85, 0.85},      /* Int. specc. */
     VERSPIEGELT,             /* Oper. luce  */
     0                        /* Tipo superf.*/
     /* User */
    }
   },
```

```
{
 KUGEL,                          /* Tipo         */
 { 0.0, 13.0, 0.0},              /* Posizione    */
 5.0,                            /* Raggio       */
 { /* Materiale: */
  {0.00, 0.00, 0.00},            /* Colore       */
  {0.00, 0.00, 0.00},            /* Colore 2     */
  {0.00, 0.00, 0.00},            /* Colore 3     */
  {0.00, 0.00, 0.00},            /* Kr           */
  { 5.0,  5.0,  5.0},            /* Fuoco f      */
  {0.85, 0.85, 0.85},            /* Int. specc.  */
  VERSPIEGELT,                   /* Oper. luce   */
  0                              /* tipo superf.*/
  /* User */
 }
},
{
 KUGEL,
 {15.0,  2.0, -2.5},
 2.0,
 { /* Materiale: */
  {1.00, 0.00, 0.00},
  {0.00, 0.00, 0.00},
  {0.00, 0.00, 0.00},
  {9.00, 9.00, 9.00},
  {60.0, 60.0, 60.0},
  {0.00, 0.00, 0.00},
  NORMAL,
  0,
  /* User */
 }
},
{
 KUGEL,
 { 8.0,  2.5, 22.5},
 2.5,
 { /* Materiale: */
  {0.00, 0.00, 1.00},
  {0.00, 0.00, 0.00},
  {0.00, 0.00, 0.00},
  {1.00, 1.00, 1.00},
  { 4.0,  4.0,  4.0},
  {0.00, 0.00, 0.00},
  NORMAL,
  0,
  /* User */
 }
},
{
 KUGEL,
 { 6.1,  1.2, 18.9},
 1.0,
 { /* Materiale: */
```

```
      {1.00, 1.00, 1.00},
      {0.00, 0.00, 0.00},
      {0.00, 0.00, 0.00},
      {9.50, 9.50, 9.50},
      {50.0, 50.0, 50.0},
      {0.00, 0.00, 0.00},
      NORMAL,
      0,
      /* User */
     }
   },
   {
    KUGEL,
    {20.5,  2.2, 6.5},
    1.5,
    { /* Materiale: */
      {1.00, 0.00, 0.00},
      {0.00, 0.00, 0.00},
      {0.00, 0.00, 0.00},
      {9.50, 9.50, 9.50},
      {50.0, 50.0, 50.0},
      {0.00, 0.00, 0.00},
      NORMAL,
      0,
      /* User */
     }
   },
   {
    KUGEL,
    {12.0,  2.7, 11.0},
    2.5,
    { /* Materiale: */
      {0.00, 0.00, 1.00},
      {0.00, 0.00, 0.00},
      {0.00, 0.00, 0.00},
      {5.00, 5.00, 5.00},
      {60.0, 60.0, 60.0},
      {0.00, 0.00, 0.00},
      NORMAL,
      0,
      /* User */
     }
   }
};

struct flaeche new_flaechen[] =
{
   { /* Parete sinistra: */
    FLAECHE,                  /* Tipo        */
    { 0.0,  0.0,-30.0},       /* 3 angoli    */
    { 0.0, 20.0,-30.0},
    { 0.0,  0.0, 30.0},
    { /* Piano, verra' inizializzato in seguito: */
      {0,0,0},{0,0,0},{0,0,0},FALSE,{0,0,0}
    },
```

```
 { /* Materiale: */
  (0.40, 0.40, 0.40),      /* Colore     */
  (1.00, 0.00, 0.00),      /* Colore 2   */
  (0.00, 0.00, 0.00),      /* Colore 3   */
  (0.70, 0.70, 0.70),      /* Kr         */
  (1.00, 1.00, 1.00),      /* Fuoco f    */
  (0.00, 0.00, 0.00),      /* Int.specc. */
  NORMAL,                  /* Oper. luce */
  0,                       /* Tipo superf.*/
  /* User: */
  {
   { /* Motivo (solo dopo impostazione di tipo di superf. = 2): */
    10,10,        /* N.ro motivi in direz. m/k */
    (0x0000, 0x0000, 0x0000, 0x001c,
     0x003c, 0x005c, 0x009c, 0x011c,
     0x021c, 0x041c, 0x0ffc, 0x101c,
     0x201c, 0x401c, 0xe03e, 0x0000)
   }
  }
 }
},
{ /* Parete posteriore (specchiata): */
 FLAECHE,                  /* Tipo       */
 { 1.3,   1.3, 29.9},      /* 3 Angoli   */
 { 1.3, 18.7, 29.9},
 (18.7,  1.3, 29.9),
 { /* Piano (verra' inizializzato in seguito: */
  (0,0,0,),(0,0,0),(0,0,0),FALSE,(0,0,0)
 },
 { /* Materiale: */
  (0.00, 0.00, 0.00),      /* Colore     */
  (0.00, 0.00, 0.00),      /* Colore 2   */
  (0.00, 0.00, 0.00),      /* Colore 3   */
  (0.00, 0.00, 0.00),      /* Kr         */
  { 1.0,  1.0,  1.0},      /* Fuoco f    */
  (0.85, 0.85, 0.85),      /* Int.specc. */
  VERSPIEGELT,             /* Oper. luce */
  0,                       /* Tipo superf.*/
  /* User */
 }
},
{ /* parete posteriore (non specchiata): */
  FLAECHE,
  { 0.0,  0.0, 30.0},
  { 0.0, 20.0, 30.0},
  (20.0,  0.0, 30.0),
  { /* Piano, verra' inizializzato in seguito: */
   (0,0,0),(0,0,0),(0,0,0),FALSE,(0,0,0)
  },
  { /* Materiale: */
   (1.00, 1.00, 1.00),
   (0.00, 0.00, 0.00),
```

```
    {0.00, 0.00, 0.00},
    {1.10, 1.10, 1.10},
    {99.0, 99.0, 99.0},
    {0.00, 0.00, 0.00},
    NORMAL,
    0,
    /* User */
   }
 },
 { /* Piano del tavolo: */
  FLAECHE,
  { 0.0,  0.0,-30.0},
  { 0.0,  0.0, 30.0},
  {20.0,  0.0,-30.0},
  { /* Piano, verra' inizializzato in seguito: */
   {0,0,0},{0,0,0},{0,0,0},FALSE,{0,0,0}
  },
  { /* Materiale: */
   {0.70, 0.00, 0.00},
   {0.00, 0.00, 0.00},
   {0.00, 0.00, 0.00},
   {1.00, 0.50, 0.50},
   { 1.0,  1.0,  1.0},
   {0.00, 0.00, 0.00},
   NORMAL,
   2,
   /* User: */
   {
    { /* Motivo: */
     5,5,        /* Numero motivi in direzione k/m */
     {0xeeee, 0xeeee, 0xeeee, 0xeeee,
      0xeeee, 0xeeee, 0xeeee, 0xeeee,
      0xeeee, 0xeeee, 0xeeee, 0xeeee,
      0xeeee, 0xeeee, 0xeeee, 0xeeee}
    }
   }
  }
 },
 { /* Fondo scatola: */
  FLAECHE,
  { 5.0,  0.1,  3.0},
  {15.0,  0.1,  3.0},
  { 5.0,  0.1, 20.0},
  { /* Piano, verra' inizializzato in seguito */
   {0,0,0},{0,0,0},{0,0,0},FALSE,{0,0,0}
  },
  { /* Materiale: */
   {1.00, 1.00, 1.00},
   {0.00, 0.00, 0.00},
   {0.00, 0.00, 0.00},
   {1.00, 1.00, 1.00},
   { 1.0,  1.0,  1.0},
   {0.00, 0.00, 0.00},
   NORMAL,
```

```
   1,
   /* User: */
   (
    ( /* Riquadri: */
     4, 4,        /* Numero riquadri in direzione m/k */
     (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
    )
   )
  )
 ),
 ( /* Coperchio scatola: */
  FLAECHE,
  (15.0,  5.0,  3.0),
  (15.0,  5.0, 20.0),
  (22.0,  0.0,  3.0),
  ( /* Piano, verra' inizializzato in seguito */
   (0,0,0),(0,0,0),(0,0,0),FALSE,(0,0,0)
  ),
  ( /* Materiale: */
   (1.00, 1.00, 1.00),
   (0.30, 0.30, 0.30),
   (0.00, 0.00, 0.00),
   (1.00, 1.00, 1.00),
   ( 1.0,  1.0,  1.0),
   (0.00, 0.00, 0.00),
   NORMAL,
   1,
   /* User: */
   (
    ( /* Riquadri: */
     4, 4,        /* Numero riquadri in direztione m/k */
     (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
    )
   )
  )
 ),
 ( /* Parete anteriore scatola: */
  FLAECHE,
  ( 5.0,  0.1,  3.0),
  ( 5.0,  5.0,  3.0),
  (15.0,  0.1,  3.0),
  ( /* Piano, verra' inizializzato in seguito */
   (0,0,0),(0,0,0),(0,0,0),FALSE,(0,0,0)
  ),
  ( /* Materiale: */
   (1.00, 0.00, 0.00),
   (0.00, 0.00, 0.00),
   (0.00, 0.00, 0.00),
   (1.00, 0.00, 0.00),
   ( 1.0,  1.0,  1.0),
   (0.00, 0.00, 0.00),
```

```
  NORMAL,
  0,
  /* User */
 }
},
{ /* Parete posteriore scatola: */
 FLAECHE,
 { 5.0,  0.1, 20.0},
 { 5.0,  5.0, 20.0},
 {15.0,  0.1, 20.0},
 { /* Piano (verra' inizializzato in seguito: */
  {0,0,0,},{0,0,0},{0,0,0},FALSE,{0,0,0}
 },

 { /* Materiale: */
  {1.00, 0.00, 0.00},
  {0.00, 0.00, 0.00},
  {0.00, 0.00, 0.00},
  {1.00, 0.00, 0.00},
  { 1.0,  1.0,  1.0},
  {0.00, 0.00, 0.00},
  NORMAL,
  0,
  /* User */
 }
},
{ /* Parete laterale destra scatola: */
 FLAECHE,
 {15.0,  0.1,  3.0},
 {15.0,  5.0,  3.0},
 {15.0,  0.1, 20.0},
 { /* Piano, verra' inizializzato in seguito: */
  {0,0,0},{0,0,0},{0,0,0},FALSE,{0,0,0}
 },
 { /* Materiale: */
  {1.00, 0.00, 0.00},
  {0.00, 0.00, 0.00},
  {0.00,. 0.00, 0.00},
  {1.00, 0.00, 0.00},
  { 1.0,  1.0,  1.0},
  {0.00, 0.00, 0.00},
  NORMAL,
  0,
  /* User */
 }
},
{ /* Parete laterale sinistra scatola: */
 FLAECHE,
```

```
    { 5.0,  0.1,  3.0},
    { 5.0,  5.0,  3.0},
    { 5.0,  0.1, 20.0},
    { /* Piano, verra' inizializzato in seguito: */
     {0,0,0},{0,0,0},{0,0,0},FALSE,{0,0,0}
    },
    { /* Materiale: */
     {1.00, 0.00, 0.00},
     {0.00, 0.00, 0.00},
     {0.00, 0.00, 0.00},
     {1.00, 0.00, 0.00},
     { 1.0,  1.0,  1.0},
     {0.00, 0.00, 0.00},
     NORMAL,
     0,
     /* User */
    }
  }
};

/* Preparazione di tutte le strutture dati: */

VOID init_welt(welt, beo)
struct welt        *welt;    /* Puntatore struttura mondo       */
struct beobachter *beo;     /* Puntatore all'osservatore       */

{

REGISTER WORD fl;

/* Approntamento strutture mondo: */
welt->anz_kugeln =
  sizeof new_kugeln / sizeof (struct kugel);
welt->kugeln = (struct kugel *)new_kugeln;
welt->anz_flaech =
  sizeof new_flaechen / sizeof (struct flaeche);
welt->flaech = (struct flaeche *)new_flaechen;
welt->anz_licht =
  sizeof new_licht / sizeof (struct lichtquelle);
welt->licht = (struct lichtquelle *)new_licht;
welt->anz_hinten = 1;
welt->hinten = (struct hintergrund *)(&new_hint);
```

```
welt->anz_obj = welt->anz_kugeln +
                welt->anz_flaech +
                welt->anz_licht + 1;

COLLET(welt->hintlicht, new_welt.hintlicht);

/* Approntamento struttura osservatore: */
*beo = new_beo;

beo->x_winkel *= PI/180;        /* Trasformazione angolo in Gradi */

/* Approntamento strutture piano in tutte strutt. di superficie */
for (fl=0; fl<welt->anz_flaech; fl++)
{
   /* Punto fisso P0 e vettori di direzione  a, b: */
   VEKLET(new_flaechen[fl].ebene.p0, new_flaechen[fl].p1);
   VEKSUB(new_flaechen[fl].ebene.a,
          new_flaechen[fl].p2, new_flaechen[fl].p1);
   VEKSUB(new_flaechen[fl].ebene.b,
          new_flaechen[fl].p3, new_flaechen[fl].p1);
}
```

Ecco che finalmente la programmazione è di nuovo divertente!

Non c'è molto da dire come commento. La cosa più importante è la partenza, poi tutto il resto va avanti da solo. L'esecuzione dura solo il tempo necessario a completare tutta l'immagine, e ciò dipende naturalmente dal numero e dimensione degli oggetti, dal numero delle fonti di luce e dal numero dei corpi a specchio, nonché naturalmente anche dal compilatore che si sta usando. La versione eseguibile presente sul dischetto è stata compilata con il compilatore Aztec, versione 3.4a e, a causa delle molte operazioni in virgola mobile (FLOAimtg point), essa è circa due/tre volte più veloce dello stesso programma compilato con il Lattice, in una vecchia versione precedente alla 3.10 (es. V 3.03). Tuttavia ci si potrà accontentare di alcune ore di tempo di calcolo, per lo scenario qui presentato, tenendo presente che il programma è piuttosto veloce; di quanti giorni o settimane necessiterebbe infatti un programma in Basic per produrre lo stesso risultato? Per fortuna i possessori del Lattice C, in versioni posteriori alla 3.10, potranno utilizzare le veloci routine FFP del sistema operativo dell'Amiga senza i problemi che avrebbero incontrato con le versioni precedenti (Ved. Cap. 2). Con ciò, le velocità di esecuzione dei due compilatori si differenziano in questo. Ma vale la pena di aspettare!. In ogni caso, se il lettore troverà l'attesa troppo lunga, potrà cominciare solo con una sfera e una fonte di luce. In tal caso l'immagine viene completata in pochi minuti.

Se, in qualunque momento si vuole riprendere il comando dell'elaboratore, sarà sufficiente premere il tasto ESC (anche durante la fase di disegno) ed il programma verrà interrotto. Diversamente sarà possibile, tramite il Mouse, posizionare lo schermo dietro un altro. Dopo aver abbassato sotto CLI la priorità di task del programma di Ray-Tracing con il comando **CHANGETASKPRI**, sarà possibile lavorare normalmente con il computer mentre il programma di Ray-Tracing funzionerà in background.

Ipotizziamo ora che l'immagine completa si trovi immobile sullo schermo, e di volerla mostrare a un amico il giorno successivo. Non è necessario lasciare acceso il computer per tale periodo; basterà premere il tasto <S> e l'immagine verrà memorizzata su disco in formato IFF. Sul dischetto qui allegato è contenuto un piccolo programma di caricamento che farà apparire in qualunque momento, su richiesta, l'immagine memorizzata. Se non si tratta di una immagine in EXTRA_HALFBRITE, non sarà un problema caricare l'immagine completa per es. in Deluxe Paint ed elaborarla in seguito.

Per la rappresentazione dell'immagine si è scelto, per la versione stampata, il modo grafico EXTRA_HALFBRITE, che rende possibile gestire contemporaneamente sullo schermo 64 colori diversi. Coloro i quali non sono in grado di rappresentare questo modo (gli Amiga 1000 non lo possiedono ancora e molti 500 hanno dei problemi) e che non possiedono un compilatore, allo scopo di modificare leggermente il programma di cui sopra e ricompilarlo, troveranno sul dischetto allegato un programma già compilato per 32 colori. In esso tuttavia la qualità dei colori sarà molto inferiore (cioè è possibile utilizzare un numero minore di tonalità di colore). Sul dischetto troveremo inoltre dei file sorgenti ed il programma eseguibile per una immagine in modo HAM (Hold-And-Modify), nel quale possono venire utilizzati, anche se con qualche limitazione, tutti i 4096 colori dell'Amiga. Solo la programmazione della Routine di posizionamento dei punti dovrà venire leggermente modificata. In tutti i modi, l'importante è una scelta intelligente dei colori nel registro di Palette, che determina esattamente ed in ultima istanza la qualità dell'intera immagine.

Non dovrebbe comunque essere un problema per nessuno impostare il modo Interlace e lavorare con doppia risoluzione. Tuttavia in questo caso il calcolo di una immagine avrà una durata doppia del normale, per cui non è consigliabile per le prime applicazioni (anche se bisogna dire che il modo Interlace produce risultati entusiasmanti). Al momento dell'impostazione del modo Interlace è necessario ricordarsi di dimezzare la dimensione verticale del punto (cioè ypg nella struttura dell'osservatore) dal momento che un punto è alto la metà della sua larghezza. Se si tralascia questo compito, l'immagine apparirà molto deformata.

Passiamo ora al programma vero e proprio. Le necessarie basi matematiche e algoritmiche sono già state apprese nelle sezioni precedenti, e costituiscono le premesse per le spiegazioni che seguono.

Cominciamo dall'inizio - Sotto il titolo "Costanti Hardware" vengono impostate alcune costanti tramite #define, che determinano il numero dei colori raggiungibili, la risoluzione Hardware ed il modo grafico selezionati. Naturalmente questi valori potranno venire modificati a piacere in qualunque momento. Come abbiamo già detto, attualmente sono impostati 64 colori con 32 registri di Palette, e con ciò abbiamo 6 bit-plane, una risoluzione di 320 × 200 in modo EXTRA_HALFBRITE. Il massimo di 16 livelli di intensità del colore è naturalmente preimpostato in modo fisso e non modificabile.

Quindi abbiamo delle definizioni ed inizializzazioni note già da tempo: strutture per un nuovo schermo, definizione ed inizializzazione di una nuova finestra, in main( ) il

programma apre qualche library, il nuovo schermo e la nuova finestra e salta quindi al programma principale vero e proprio do_program( ), di cui ci occuperemo fra poco.

Prima però abbiamo a che vedere ancora con alcuni #define. Qui vengono definite soprattutto delle macro per le quali non dovrebbe venire approntato un sotto-programma separato per motivi di velocità (se non si conoscono le macro in C, verificare nel testo sul C di cui si dispone gli argomenti "Preprocessore" o "#define"): si tratta della definizione di massimo, minimo, valore assoluto ed elevazione a potenza, nonché di numerose operazioni vettoriali necessarie. Nel programma un vettore viene sempre considerato come una matrice di tre elementi. Ogni elemento rappresenta una coordinata del vettore (es.: v[0], v[1] e v[2] sono i tre elementi vx, vy e vz del vettore v).

Se ora si devono sommare due vettori $\vec{a}$ e $\vec{b}$ e li si deve memorizzare come $\vec{c}(\vec{c} = \vec{a} + \vec{b})$, il programma dovrà sommare, conformemente alla definizione dell'addizione dei vettori, che dovrebbe già esserci familiare, ogni singolo elemento di $\vec{a}$ ad ogni suo corrispondente elemento di $\vec{b}$ e memorizzarlo come elemento di c:

```
c[0] = a[0] + b[0]
c[1] = a[1] + b[1]
c[2] = a[2] + b[2]
```

Corrisponderà quindi a:

$$\begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$$

Lo stesso vale per tutte le altre operazioni con vettori. Le diverse macro (per l'addizione, la macro VEKADD(c,a,b)) eseguiranno esattamente tali operazioni. Come parametri di macro dovranno venire indicati in seguito, nel programma solo i nomi delle matrici, es. VEKADD(risultato, vettore1, vettore2) sommerà gli elementi dei vettori vettore 1[ ] e vettore 2[ ] e memorizzerà il risultato in [ ]. Fare attenzione al fatto che è possibile incorporare solo poche macro in grandi espressioni, diversamente molte di esse daranno come valore di risultato VOID.

L'importante passo successivo è la dichiarazione di tutte le strutture impiegate nel programma, che costituiscono in pratica l'ossatura dell'intero sistema. In linea di massima (quasi) tutti i dati vengono raccolti in strutture che vengono trasferite come parametri di comunicazione di funzione in funzione. In realtà non vengono trasferite le strutture vere e proprie, bensì solo il loro puntatore. In tal modo possono venire resi accessibili molto velocemente ad ogni struttura molti dati locali. Analizziamo quindi le strutture una per una:

struct punkt (strutt punto):
Si tratta di una pura struttura di comunicazione che contiene dati che vengono calcolati una sola volta nel corso del programma e che forse serviranno in seguito. Di con-

seguenza il calcolo di un punto di intersezione, per esempio, non è necessario nelle diverse routine. Una volta determinato, esso sarà disponibile immediatamente a tutte le altre funzioni. Anche altre strutture contengono tali "memorie di valore intemedio" che verranno completate in seguito durante il programma. In ogni caso, qui abbiamo a che fare con tutti i dati che descrivono un punto nello spazio: posizione, colore, valore s per l'equazione vettoriale della retta del raggio dello sguardo ecc. Alcune memorie (user 0-2) sono parzialmente ancora inutilizzate e previste per futuri ampliamenti. E' in esse che si trova infine il colore del punto dello schermo.

struct gerade (strutt retta):
Una difinizione vettoriale di una retta è composta da un punto fisso P0 e da un cosiddetto vettore di direzione $\vec{a}$ (ved. equazione vettoriale di una retta). Ogni retta del programma viene determinata in questo modo e con questa struttura.

struct ebene (strutt piano):
Conformemente all'equazione vettoriale k-m (oppure equazione s-t), un piano è determinato da un punto fisso P0 e da due vettori di direzione $\vec{a}$ e $\vec{b}$. Esiste eventualmente la possibilità di aggiungere anche il vettore della normale al piano (ved. cap. 4.3), cosa che rende più semplici determinati calcoli. Questi parametri vengono trattenuti in maniera fissa dalla struttura del piano. Dal momento che il vettore della normale non è sempre necessario, il flag BOOLEANO (vero falso) n_valid determina se esso è indicato o no nella struttura in quel momento.

struct material (strutt materiale):
Arriviamo alle parti interessanti del programma. La struttura in questione determina le caratteristiche specifiche di un oggetto nello spazio. Naturalmente essa non è mai disponibile da sola, ma sarà parte di un'altra struttura, cioè di quella dell'oggetto (es. per una sfera o una superficie). E' da qui che il programma determina quale colore e quali caratteristiche di riflesso sono possedute dall'oggetto, se è a specchio (l'opzione "trasparente" non è implementata in questa versione del programma) o se la sua superficie è strutturata in maniera particolare. In linea di massima questo è il luogo per la maggior parte delle modifiche di un oggetto (lucido, opaco o metallico, chiaro, scuro, rosso, verde o bianco.).

Questa struttura è stata concepita nel modo più aperto possibile. nel caso in cui si volesse ampliare questo programma, essa offrirà spazio per molti parametri, per descrivere l'oggetto in maniera sempre più particolareggiata. Come struttura di superficie per i piani sono già stati implementati i riquadri e altri motivi a piacere. La definizione del motivo si trova all'interno della union nella struttura "muster" (motivo) (ved. sotto). Ritorneremo in seguito su queste strutture e le approfondiremo.

struct hintergrund (strutt sfondo):
Siamo giunti alla prima struttura di oggetto: lo sfondo. Normalmente una struttura di oggetto è costituita, grossolanamente, come segue:

Carattere di identificazione del tipo di oggetto (es. "0" per sfondo)
Posizione dell'oggetto
Altri parametri descrittivi
Proprietà del materiale (di solito struct material)
Dove non è presente nessun oggetto, apparirà lo sfondo. Attualmente l'unico parametro modificabile è il colore. C'è spazio per un altro colore (es. cambiamento di colore dall'orizzonte allo zenith ecc.) che però non è utilizzato in questo programma.

struct lichtquelle (strutt fonteluce):
Anche questa è una struttura di oggetto particolare. Essa definisce la posizione, luminosità e colore di una fonte di luce. Quando una fonte di luce è invisibile, essa viene trattata esattamente come una sfera monocolore, ecco perché è necessario indicarne il raggio. La costante della distanza dist_konst è la stessa costante kd che abbiamo già incontrato trattando delle riflessioni diffuse e a specchio. Essa serve a non far apparire troppo scure delle fondi di luce molto lontane (kd viene moltiplicata per la distanza d della fonte di luce rispetto all'oggetto in questione). Minore è kd, più chiara sarà la fonte di luce.

struct kugel (strutt sfera):
Definizione di una sfera con tipo, posizione, raggio e le diverse caratteristiche di materiale.

struct flaeche (strutt superficie):
Un parallelogramma definito tramite tre punti angolari e le costanti del materiale. La struttura dei piani all'interno di questa struttura verrà inizializzata solo in seguito. Quindi il programma calcola la formula del piano dai tre punti angolari.

struct beobachter (strutt osservatore):
Lasciamo gli oggetti dietro di noi e rivolgiamoci all'osservatore, la cui posizione, direzione dello sguardo ecc. ci dovrebbero già essere note. Dovrà venire indicato quanto segue:

— Posizione
— Punto al quale l'osservatore sta guardando. Con ciò, si potrà indicare direttamente un oggetto.
— In alternativa, vettore dello sguardo, cioè direzione dello sguardo.
— Distanza dalla lastra di vetro (normalmente uguale a 1, ma può assumere anche altri valori, per cui l'angolo di visuale verrà incrementato o diminuito proporzionalmente). Esso determina, per così dire, l'ampiezza di messa a fuoco dell'obiettivo.
— Larghezza e altezza di un punto sullo schermo o sull'apparecchiatura sulla quale si vuol fare uscire il grafico, in unità a piacere (xpg ed ypg).
— Numero dei punti da tracciare in direzione x ed y.
— Angolo di visuale orizzontale. Con esso si determina la sezione di immagine che si vuol vedere (è un altro modo per indicare l'ampiezza di messa a fuoco dell'obiettivo.

Gli altri parametri verranno calcolati autonomamente in seguito dal programma, partendo dai valori indicati. Ancora una cosa relativamente al punto cui si guarda cioè al vettore dello sguardo: sarà necessario dichiarare chiaramente al computer, in qualunque maniera, in quale direzione sta guardando l'osservatore (verso il basso, l'alto, a destra o a sinistra ecc.). Nelle nostre derivazioni matematiche siamo partiti da un determinato punto al quale egli stava guardando. Con ciò siamo in grado di evitare di "guardare oltre" un determinato oggetto. Dal momento però che con questo tipo di indicazione cambia la direzione dello sguardo, e quindi la prospettiva, non appena si sposta l'osservatore, spesso è più vantaggioso indicare la direzione dello sguardo come vettore dello sguardo (calcolabile dalla differenza fra il punto cui si guarda e la posizione dell'osservatore, ved. sopra). Quindi possiamo scegliere fra l'utilizzo del punto cui si guarda come indicazione di direzione, per cui imposteremo a zero il vettore dello sguardo, oppure l'indicazione del vettore dello sguardo. Il programma comprende e accetta ambedue i casi.

Con l'indicazione della risoluzione (cioè del numero dei punti da tracciare in direzione x ed y) determiniamo quanto sarà grande l'immagine sullo schermo. Una piccola immagine viene naturalmente calcolata molto più velocemente ed è più adatta a fungere da test che non una a tutto schermo, per la quale il computer lavorerà eventualmente diverse ore.

struct welt (strutt mondo):
L'ultima struttura tiene insieme tutto il mondo. E' in questa struttura che il computer legge quanti oggetti, quante fonti di luce, sfere e superfici si trovano nel mondo e dove trovare le definizioni. Tutte le sfere (nonché tutte le fonti di luce e superfici) sono memorizzate per esempio in una matrice composta da strutture di sfere. L'indirizzo di partenza di tale matrice si trova qui. Qui si indica anche quanto deve essere intensa la luce diffusa, cioè la luminosità di sottofondo, cioè quanto deve essere chiaro un oggetto che si trova in ombra.

A questo punto abbiamo ricevuto una visione d'insieme delle strutture di questo mondo sintetico. Ora entriamoci.

Il fulcro di tutti i calcoli del computer, nonché di tutti i voli e cadute del programmatore è la routine principale do_program( ). E' qui che si trovano i due loop FOR nidificati uno nell'altro, che scandiscono ogni punto dello schermo, ne calcolano il colore e lo fanno apparire sul monitor.

Tuttavia, prima di essa, il programmatore ha posto le inizializzazioni, che si manifestano nelle routine init_welt( ), init_farben( ) e get_konst( ) che dobbiamo assolutamente analizzare.

init_welt( ) dispone tutte le strutture nel modo in cui il programma vuole trovarle. Essenzialmente essa recupera i dati che noi abbiamo immesso solo alla fine per i singoli oggetti, fonti di luce, osservatore ecc. Torneremo in seguito su questa routine.

Continuiamo quindi con **init _ farben( ), che prepara la Palette di colori per la funzione di posizionamento del punto e che è quindi secondaria. Lasciamo da parte per un attimo anche questa routine.**

Passiamo ora a get _ konst( ): il punto principale è costituito da due esercizi obbligatori che hanno a che vedere con la struttura dell'osservatore: il calcolo del vettore dello sguardo s dall'osservatore alla lastra di vetro, che viene calcolato dal punto cui l'osservatore guarda o, se disponibile, dal vettore della vista. Qui utilizziamo per la prima volta le macro #define: VEKLET( ), VEKSUB( ), VEK _ LAENGE( ) e MULVEK( ). Cerchiamo di chiarircene bene il funzionamento fin da subito: esse ci serviranno molto in seguito. Il risultato, il vettore dello sguardo, memorizza quindi la routine negli spazi non ancora occupati della struttura dell'osservatore (beo è in questo caso un puntatore alla struttura dell'osservatore). Quindi si accederà ad un elemento:

(* beo).blickv[0]

Oppure, più brevemente

beo -> blickv [0].

Per quanto concerne il calcolo dei vettori dei passi $\vec{xe}$ ed $\vec{ye}$, non è necessario entrare nei dettagli, in quanto sono già stati affrontati in precedenza. Essi vengono ora moltiplicati per la singola dimensione di punto (xpg, ypg) al fine di determinare semplicemente tramite la formula seguente un punto Pm sulla lastra di vetro, avente le coordinate x,y:

$$P_m = B + \vec{s} + x^*(xpg^*\vec{xe}) + y^*(xpg^*\vec{ye})$$

Ciò accade anche nella funzione get _ punkt( ).

Con ciò siamo già dentro i due loop di FOR di **do _ program( )**. Dopo che le coordinate spaziali del punto sulla lastra di vetro sono note, non dovrebbe essere difficile calcolare il raggio dello sguardo per il quale si sta controllando se interseca un oggetto o no. A questo punto incontriamo un'altra macro GET _ GERADE( ) che determina l'equazione di una retta dati due punti e la scrive nella struttura di retta del raggio dello sguardo.

Di seguito troviamo le inizializzazioni del cosiddetto coefficiente di intensità, che viene utilizzato esclusivamente per la limitazione del numero di specchiature multiple e del valore del colore del punto, che deve ancora venire determinato.

A questo punto inizia l'inseguimento del raggio: **ray _ trace( )**. Questa routine determina, più o meno brevemente, il colore esatto del punto sulla lastra di vetro, con una precisione di sette cifre decimali, nel campo che va da 0 a 1. Il colore viene determinato dai valori di intensità di ciascuno dei tre colori di base rosso, verde e blu. Questo valore di colore determinato (che si trova nella struttura del punto) viene trasmesso dal do _ programm( ) alla routine **plot( )**, che si occupa di far apparire un punto sullo schermo avente circa quel colore (con un massimo di 64 colori è già un problema). Nel caso in cui siamo riusciti a concludere con successo anche questa azione, la routine controlla se un tasto è stato premuto (infatti è possibile interrompere il tutto tramite la pressione del tasto <Esc>), quindi passa al punto successivo.

ray_trace( ):
ray_trace( ) è certamente una delle funzioni più importanti di tutto il programma, dal momento che essa insegue un raggio, calcolando tutti i punti di intersezione di questo raggio con tutti gli oggetti presenti. Essa seleziona il punto di intersezione successivo (rispetto al punto fisso del raggio) e determina il colore di tale punto dell'oggetto. Essa è molto importante anche perché può venire chiamata ricorsivamente in caso di oggetti specchiati.

Come molte funzioni centrali, anche **ray_trace( )** è piuttosto piccola e trascurabile, dal momento che funge principalmente solo come "punto di smistamento", in quanto essa affida l'esecuzione dei compiti veri e propri ed i "dettagli" ad altre routine da lei chiamate. Quindi passa alla determinazione del punto di intersezione successivo. Questo compito viene svolto dalla funzione **schnitt_objs( )**, la quale trasferisce i tipi di oggetto dell'oggetto successivo ed il valore s dell'equazione della retta sulla quale si trova il punto di intersezione. Quindi non viene necessariamente calcolato il punto di intersezione vero e proprio sotto forma di coordinate, bensì sotto forma di parametro s (fattore di allungamento) per l'equazione vettoriale della retta (a partire dal quale è naturalmente possibile determinare in qualunque momento il punto stesso). Con queste informazioni, ray_trace( ) potrà quindi saltare direttamente alle routine che determinano il colore dell'oggetto in questo punto: hintergrundfarbe( ), lichtfarbe( ), kugelfarbe( ), flaechenfarbe( ) (colore sfonro, colore luce, colore sfera, colore superficie).

schnitt_objs( ) (taglia ogg):
Il nucleo del calcolo del punto di intersezione schnitt_objs( ) taglia ogg è naturalmente un punto di smistamento per le singole routine, molto diverse tra loro, che determinano il punto di intersezione di una sfera o di una superficie ecc. con il raggio della vista: schnitt_kugel_a( ) (taglia sfera), schnitt_flaech( ) (taglia superficie). Per ogni tipo di oggetto è stato preparato un loop di FOR che esamina tutti gli oggetti memorizzati nella matrice degli oggetti. Nel caso in cui esista un punto di intersezione con il singolo oggetto, esso controllerà se tale punto di intersezione si trova più vicino all'osservatore del punto di intersezione successivo. In tal caso, la routine memorizzerà la s e l'indirizzo della singola struttura di oggetto (in realtà il puntatore alla prima WORD della struttura di oggetto, cioè il tipo di oggetto) e proseguirà con l'oggetto successivo. In caso di superfici, vengono registrate anche le coordinate del punto di intersezione disponibili già calcolate, nonché i valori m e k dell'equazione di un piano (cioè la posizione del punto di intersezione sul piano), di cui si potrà avere bisogno in seguito (non è infatti necessario fare i calcoli due volte).

Dopo che tutti gli oggetti presenti sono stati controllati in questa maniera e dopo che è stato determinato ogni singolo punto di intersezione, dovremo bene o male occuparci dello sfondo. Con la seguente istruzione

```
return(**objekt);
```

la routine restituisce il tipo di oggetto successivo (che si trova in ogni struttura di oggetto), memorizza velocemente s nella struttura del punto e termina.

Calcolo del punto di intersezione:

schnitt_kugel_a( ) (taglia sfera):
A questo proposito non abbiamo molto da aggiungere, dal momento che è già stato quasi tutto spiegato nelle dissertazioni teoriche. La routine calcola il (i) punto (punti) di intersezione tra il raggio trasferito ed una sfera indicata, controllando contemporaneamente se il più piccolo s trovato (esistono sempre due punti di intersezione, se tralasciamo il caso della tangenza) è maggiore di 0 (più esattamente: maggiore di quasi 0, a causa della imprecisione di calcolo). In caso contrario, la routine calcolerà l's maggiore come punto di intersezione (ciò significa che l'osservatore si trova all'interno della sfera).

schnitt_flaech( ) (taglia superficie):
Anche in questo caso non c'è molto da spiegare, in quanto abbiamo appena affrontato le basi matematiche. Osserviamo comunque un punto all'inizio della routine. In esso il programma controlla se il vettore della normale al piano, necessario in questo caso, è già stato registrato nella struttura di piano della superficie. In caso contrario il programma lo calcola velocemente tramite un prodotto vettoriale, lo memorizza nella struttura del piano, lasciandolo a disposizione per futuri usi come valido (n_valid). Quindi anche per questa superficie non sarà più necessario determinare il vettore della normale.

Torniamo un attimo alla funzione **ray_trace( )**. Abbiamo già superato il calcolo del punto di intersezione, l'oggetto più vicino è stato determinato e si trova in **obj_typ**. A questo punto dobbiamo determinare il colore del punto di intersezione con l'oggetto. A seconda di quale oggetto si tratti, la routine salterà, grazie all'istruzione SWITCH alle routine di calcolo del colore dei singoli tipi di oggetto:

Calcoli del colore:

Per quanto concerne i calcoli del colore dei singoli oggetti, possiamo dare libero sfogo alla nostra fantasia: potremo infatti inserire ciò che più ci piace per quanto concerne le strutture di superficie, l'andamento dei colori o altri "effetti speciali".

hintergrundfarbe( ) (colore sfondo):
Questa determinazione è molto semplice, in quanto il colore dello sfondo dovrà venire preso semplicemente dalla struttura dello sfondo. In questo punto è possibile anche programmare un andamento sfumato del colore o simile. Dal momento che il calcolo del colore dello sfondo ha luogo molto velocemente, i singoli passaggi, nei quali non è ancora possibile vedere nulla, sono altrettanto veloci (potremo verificarlo direttamente sullo schermo).

lichtfarbe( ) (colore luce):
Anche in questo caso non incontreremo nessun problema, dal momento che nella luce non saranno presenti specchiature, deformazioni o effetti simili.

kugelfarbe( ) (colore sfera):
Nel caso in cui il raggio incontri una sfera, diventa piuttosto laborioso riuscire a crea-

re una bella immagine. Vedremo quindi per la prima volta come utilizzare le formule per la riflessione diffusa e a specchio. A tale scopo abbiamo bisogno da un lato delle coordinate del punto di intersezione, dall'altro del vettore della normale, cioè del vettore perpendicolare alla superficie alla superficie della sfera. L'Amiga calcola coerentemente le parti necessarie: il punto di intersezione dalla equazione della retta del raggio e dal fattore di allungamento s (dal calcolo del punto di intersezione) e la normale come vettore dal centro della sfera al punto di intersezione.

Inoltre esso determinerà il colore di base dell'oggetto, che dovrà venire modificato secondo le formule di riflessione. Con questo bagaglio, la routine kugelfarbe( ) avvia la funzione centrale per il calcolo dell'intensità del punto farbintens( ). Dal momento che questa routine non dipende da valori specifici dell'oggetto (le sue dimensioni di riferimento sono solo il punto di intersezione, la normale, il colore del punto e le costanti del materiale neutrali) essa non ha alcun collegamento con i singoli oggetti (ad esclusione di fonti di luce e sfondo).

flaechenfarbe( ) (colore superficie):
Lo scopo di questa routine è ovviamente identico a quello della routine appena incontrata: punto di intersezione, normale, colore del punto. Ciononostante essa è un po' più lunga, cosa che dipende dai molti effetti di superficie addizionali che possono venire selezionati per tali superfici: in tinta unita, a scacchiera oppure bicolore con un motivo a piacere.

Il punto di intersezione del raggio con la superficie è già noto! Sappiamo infatti che è già stato determinato una volta, e che da allora è a nostra disposizione, un lavoro in meno. Anche il vettore della normale non rappresenta più un problema, essendo anch'esso già stato determinato (si trova ora nella struttura del piano della superficie).

Ciò che resta da determinare è il colore di fondo dell'oggetto. Il programmatore ha differenziato, a questo scopo, tre casi diversi tramite SWITCH. Nella struttura del materiale esiste infatti un byte con il nome oberf_typ. E' in esso che metteremo il numero corrispondente al tipo della superficie desiderata:

oberf_typ = 0: (tipo superficie = 0)
Questo è il caso più semplice e più noto: la routine si procura il colore di base dell'oggetto direttamente dall'istruzione del colore della struttura del materiale, cioè l'oggetto è in tinta unita.

oberf_typ = 1: (tipo superficie = 1)
L'inserimento di un uno in questo byte indica che l'oggetto (in questo caso la superficie) deve essere a quadri. I due colori per i riquadri provengono dai primi due registri di colore della struttura del materiale. In aggiunta a ciò, forniamo altri due valori alla struttura del motivo all'interno della struttura del materiale: il numero dei riquadri in direzione x e di quelli in direzione y rispetto al punto 0 della superficie (P0), in altre parole in direzione m e k, se vogliamo tralasciare l'equazione vettoriale del piano.

zzare le formule
da un lato delle
e, cioè del vetto-
calcola coerente-
a retta del raggio
e la normale co-

venire modificato
ugelfarbe( ) avvia
( ). Dal momento
ue dimensioni di
punto e le costan-
oggetti (ad esclu-

outine appena in-
stante essa è un
nali che possono
e bicolore con un

opiamo infatti che
sizione, un lavoro
oblema, essendo
della superficie).

rogrammatore ha
struttura del mate-
tteremo il numero

colore di base del-
ateriale, cioè l'og-

esto caso la super-
ai primi due registri
altri due valori alla
dei riquadri in di-
(P0), in altre parole
del piano.

A questo punto dobbiamo ancora determinare in quale riquadro
colore 2) si trova il punto, e con ciò ne avremo ottenuto il colore.

oberf _ typ = 2: (tipo superficie = 2)
Con questa impostazione possiamo definire da soli un motivo di
fare apparire sull'oggetto delle righe, dei punti o anche una piccola
so tempo imposteremo anche il numero di volte per le quali il motivo
tuto sulla superficie (anche qui in direzione m e k). Anche in questo
a disposizione provengono dai primi due registri di colore.

farbintens( ): (intensità colore)
E' qui che si incontrano di nuovo tutte le routine che devono det
di un oggetto o di un punto di un oggetto. farbintens( ) necessita d
vettore della normale, del raggio di luce, della struttura del punto (
del punto e il coefficiente di intensità), della struttura del materiale de
ralmente della struttura del mondo (tutto in puntatori).

Ora il programma deve prima di tutto decidere se l'oggetto è a
no. In caso positivo (determinato dalla variabile **licht _ typ** nella strutt
verrà chiamata la routine di specchio. Il suo effetto finale è quello di
di riflessione (angolo di incidenza uguale all'angolo di uscita, come
visto nelle nostre considerazioni matematiche) e l'avviamento di un
quale viene richiamata la routine **ray _ trace( )** con il raggio di rifless
della vista. Ciò significa quindi che si andrà alla ricerca dell'oggetto
cino, il quale a sua volta potrà essere a specchio oppure no. Natural
sibile procedere in questo modo all'infinito. Ipotizziamo infatti di av
a specchio che corrano parallele l'una all'altra. Ogni specchio si spec
nito nell'altro, ed il nostro programma entrerebbe in loop.

Abbiamo quindi bisogno di un criterio di interruzione per questa ri
detto coefficiente di intensità. Per ogni oggetto specchiato dovremo
numero relativo all'intensità di specchiatura (spiegel _ int[ ], da 0 a 1)
materiale. Tale numero indica la percentuale della luce che viene sp
mente dalla superficie a specchio (nessuno specchio infatti riflette
la luce che lo colpisce). Questo numero (vale anche per RGB, infat
grammare anche degli specchi "a colori") viene moltiplicato dal prog
ficiente di intensità attuale, diventando sempre più piccolo da
specchiatura (eccettuato il caso di quando lo specchio riflette al 1
di più). Non appena questo valore scende al di sotto di un determi
termina anche la ricorsione, dal momento che non c'è più luce suffic
chiatura.

Al fine tuttavia di mantenere al di sotto di un determinato valore il n
sioni (al fine di non superare le capacità dello stack, con conseguent
programma) il programma conterà contemporaneamente il numero d
variabile globale rekur _ zaehl. Se questa è maggiore di REKUR _ M
massimo permesso di ricorsioni (qui 7), si avrà comunque una inte

Fare attenzione al fatto che, per quanto concerne le **ricorsioni**, è stata preparata una struttura di punto locale completamente nuova ed una struttura diretta locale altrettanto nuova nella routine **farbintens( )**. Infatti le vecchie strutture non dovranno venire modificate nemmeno dalla recursione.

Ipotizziamo che la routine **ray_trace( )**, chiamata ricorsivamente, ritorni al proprio posto senza problemi; saremo certi che essa ha determinato il colore del punto sull'oggetto successivo e lo ha depositato nella struttura del punto. Questa (nuova) struttura del punto non è più necessaria. E' solo il valore del colore determinato che dovrà venire inserito in quella vecchia, al fine di poter procedere nei calcoli con esso. Ciò accade con la macro COLLET.

In questa posizione si incontrano di nuovo i due casi (a specchio o non a specchio). Ora comincia infatti il calcolo della riflessione specchiata e diffusa. Nel fare ciò, il programma tiene conto in un grande loop della luce che ricada eventualmente sull'oggetto da tutte le fonti di luce. Sarà però dapprima necessario verificare se il punto da calcolarsi viene illuminato o no da ciascuna delle fonti di luce. Può accadere infatti che esista un altro oggetto fra il punto e la fonte di luce e che il punto in questione si trovi veramente nell'oscurità.

Dovremo di nuovo calcolare un nuovo raggio, questa volta dal punto alla fonte di luce (la fonte di luce viene idealizzata in questa sede, per semplicità, come costituita da un solo punto, anche se possiede un raggio). Ci stavamo chiedendo se esiste un oggetto che si trovi tra il punto e la fonte di luce, in altre parole, se esiste un oggetto che venga intersecato dal raggio diretto alla fonte di luce.

La risposta a questa domanda ci viene fornita dalla routine globale per il calcolo del punto di intersezione successivo **schnitt_obj( )**, che è già stata descritta in precedenza. Il risultato di questa funzione verrà utilizzato questa volta in maniera diversa. Infatti la routine determinerà in ogni caso almeno un punto di intersezione, cioè quello con la fonte di luce vera e propria. Se la routine quindi trova proprio la fonte di luce come oggetto di intersezione più vicino, la questione è risolta e il punto non si trova nell'oscurità, per cui può avere inizio il calcolo della formula di riflessione. Se invece la routine schnitt_obj( ) trova un altro oggetto (cosa determinabile dall'indirizzo dell'oggetto che ci viene segnalato), il punto si troverà nell'oscurità, cioè non riceverà nessuna luce dalla fonte di luce in questione.

La formula per il calcolo dei due tipi di riflessione ci è già nota. Naturalmente essa dovrà venire applicata separatamente per ciascuno dei tre colori di base rosso, verde e blu.

Il programma funziona in questo modo per tutte le fonti di luce. Resta solo la luminosità di sottofondo (luce diffusa) Ih*Koh. Essa ha effetto su ciascun punto e dovrà quindi venire aggiunta alla fine. Il colore del punto è completamente determinato e la routine termina.

plot( ):

Qui ci troviamo di nuovo nel vecchio loop principale di **do_programm( )**. **ray_trace( )** è già superato, e siamo di fronte al coronamento di tutti questi calcoli complicati: plot( ), cioè l'output del punto. E' questo output che ci ha preoccupato a lungo, dal momento che eravamo costretti a lavorare con i pochi registri di colore disponibili, anche se il colore del punto può venire calcolato effettivamente con una precisione di sette posizioni decimali, abbiamo dovuto tenere conto di numerose ombre sullo stesso colore e ciononostante dovevamo produrre una immagine il più vicino possibile alla realtà. Non nascondo che la soluzione trovata in questa sede non sia necessariamente la migliore. Questa routine inoltre è anche l'unica che dovrà venire modificata in caso di trasposizione del programma su di un altro computer (eventualmente insieme a init_farben( )). Tutte le altre sono state praticamente concepite per un computer ideale con molti milioni di colori (contemporaneamente) e con una risoluzione grafica a piacere.

A questo punto dovremo bene o male spendere una parola sulla memorizzazione dei diversi colori, cosa che non abbiamo fatto al momento della presentazione di **init_farben( )**. Nella parte dati del programma, cioè circa alla fine, abbiamo la possibilità di effettuare l'assegnazione dei colori ai registri di Palette. A tale scopo forniamo per ogni colore di sfondo dei valori da 0 a 15. In init_farben( ) il programma memorizza questi valori nei registri di Palette dell'elaboratore e moltiplica i valori per $2^{16} = 65536$. Ciò ha lo scopo di memorizzare in una variabile intera (INT, a 16 bit), per motivi di velocità, anche le posizioni decimali dei valori del colore. Contemporaneamente, in init_farben( ), vengono determinati i rispettivi valori di colore per i colori da 32 a 63, per il caso nel quale si sia selezionato il modo EXTRA_HALFBRITE. Questo formato è naturalmente solo quello interno al programma, che dovrà venire calcolato all'indietro per l'adattamento all'Amiga. A dire il vero, ciò può sembrare piuttosto complicato e laborioso, ma i tentativi con numeri reali (o in virgola mobile, FLOAT) causerebbero un notevole rallentamento del programma.

Osserviamo quindi **plot( )**. In esso troviamo in x ed y la coordinata dello schermo alla quale deve venire posto il punto (per la quale l'origine si trova in alto a sinistra). Quindi la struttura di punto viene trasmessa alla funzione, ed è da essa che viene "estratto" il colore esatto del punto. In una prima fase, questa deve venire tenuta ad un valore fra 0 e 1 per R, G e B, dal momento che non può assolutamente accadere che vengano determinati valori maggiori in questa fase. Contemporaneamente, la routine trasforma i valori in virgola mobile nel formato di colore precedentemente descritto. Vediamo quindi che non vengono dimenticate nemmeno le posizioni decimali.

Il passo seguente è un po' più complesso. In esso si ricerca infatti il registro di Palette che si avvicina più degli altri al colore del punto. A tale scopo l'elaboratore esamina tutti i colori disponibili (32 o 64) ed individua i due registri per i quali la differenza fra il colore del registro ed il colore reale del punto è la minore (**reg_min**) o è la seconda in una graduatoria delle minori (**zweit_reg_min**).

Normalmente diremo: reg_min è il registro che contiene il colore che si avvicina di più al colore del punto, per cui il punto da tracciare dovrà ricevere il colore da

Normalmente diremo: reg_min è il registro che contiene il colore che si avvicina di più al colore del punto, per cui il punto da tracciare dovrà ricevere il colore da reg_min. In teoria avremmo ragione, ma verificheremo ben presto, se andiamo a controllare di persona, che ogni oggetto presenta delle fasce molto sgradevoli, che si formano nel punto in cui un registro di colore viene cambiato in un altro. Ciò è dovuto al fatto che il nostro occhio non è in grado di riconoscere chiaramente la differenza fra la intensità nr. 13 di rosso e l'intensità nr. 12 di rosso.

La soluzione a questo effetto sgradevole delle zone di confine è quella di "sfrangiare" i campi con colore rosso 12 e rosso 13, per rendere dolce il passaggio da uno all'altro. Ciò può venire ottenuto con una piccola funzione casuale zufall( ). Ad ogni sua chiamata, essa produce un numero (apparentemente) casuale fra 0 e 256 ($00-$FF). L'entità di questo numero determina se il punto deve venire tracciato con rosso 12 o rosso 13. Quanto più ci siamo spinti nel campo di rosso 13, tanto più diminuiranno le probabilità per rosso 12, e viceversa. Ciò viene ottenuto con la seguente piccola formula nell'istruzione IF:

$$gz = z + \frac{zs * mz}{(zs-es)} * wk$$

es = differenza colore fra il colore più adeguato e il colore esatto del punto
zs = differenza colore fra il secondo colore più adeguato e il colore esatto del punto
mz = entità massima del numero casuale (in questo caso 256)
wk = correzione di pesatura
z = numero casuale (da 0 a mz)
gz = numero casuale pesato (da 0 a 2 * mz)

"es" e "zs" sono già stati determinati dalla routine con il registro migliore e con quello immediatamente successivo. mz è uguale a 256 (più propriamente, uguale a 255) che è il numero casuale massimo permesso. z è il numero casuale stesso, che viene fornito come è noto da **zufall( )**. Si è introdotta anche la "correzione di pesatura" wk, al fine di spostare la probabilità di un colore nella direzione del primo colore. wk determina anche l'ampiezza del campo "sfrangiato". gz è in conclusione il numero casuale pesato tramite la formula, che si stava ricercando, che dovrà trovarsi tra 0 e 512.

Se gz è maggiore di 255, verrà scelto il colore più vicino, se gz è < =255, verrà utilizzato il secondo colore più vicino. Il risultato sarà una immagine a punti, cosa che è tuttavia quasi inevitabile (se non aumentando la risoluzione, per es. tramite Interlace).

In Fig. 6.9 è possibile rivedere uno schema generale del programma di Ray-Tracing.

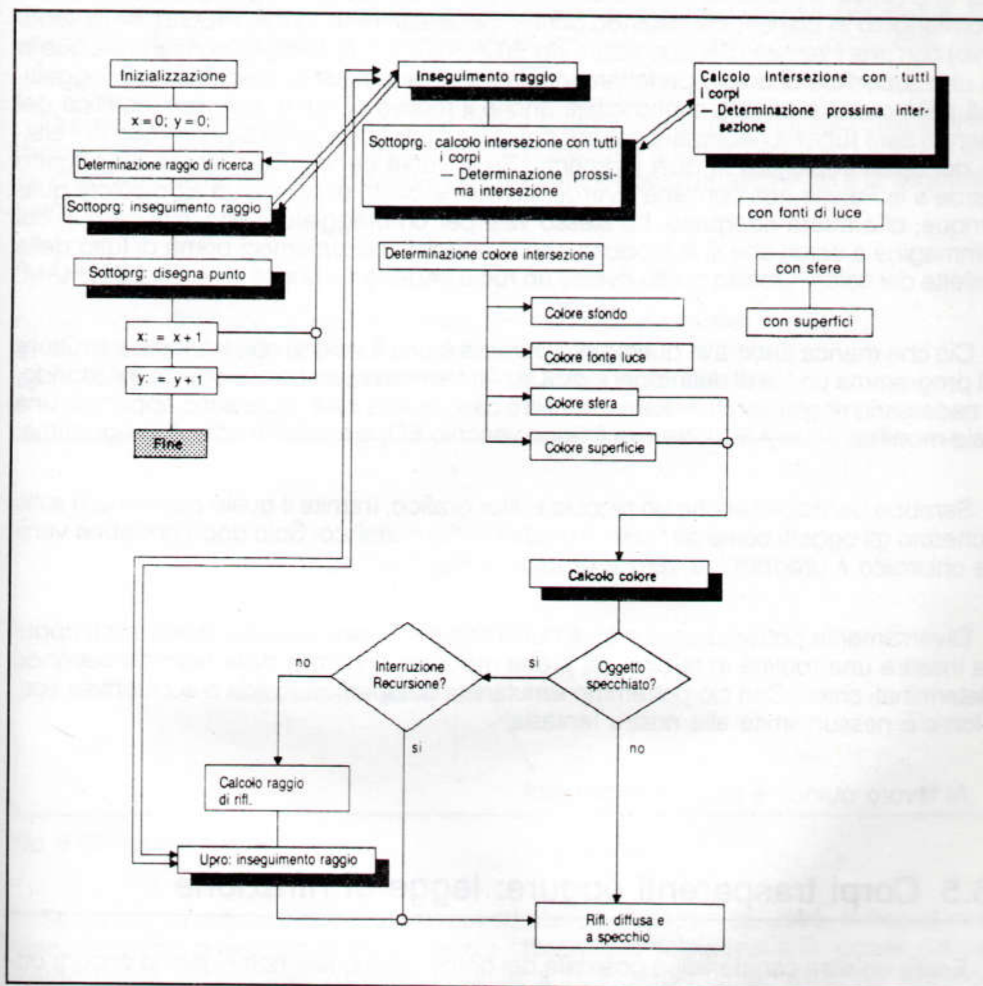


Fig. 6.9 Diagramma di flusso di un programma di RayTracing

Cosa resta? Naturalmente la modifica del mondo, cosa altro? Nel nostro mondo è molto più semplice che non nella realtà. Osserviamo i campi dati alla fine del programma, nei quali troviamo tutte le informazioni necessarie per lo scenario, cominciando con i colori di Palette, attraverso i valori dell'osservatore fino alle fonti di luce, sfere e superfici. La funzione dei diversi parametri dovrebbe già esserci nota dalle analisi teoriche svolte in precedenza. Resta da provare quali effetti risultino nella pratica dalle più svariate combinazioni. Con l'andare del tempo ne acquisiremo anche la sensazione.

Cominciamo dagli oggetti più semplici. Nel caso in cui si avesse invece in mente di tracciare uno scenario completo, sarà meglio munirsi di carta e penna. Consiglio per una prova: modifichiamo il colore 1 del piano del tavolo a righe gialle e nere, trasformandolo in bianco, impostando contemporaneamente "VERSPIEGELT" (a specchio) con una intensità di specchiatura del 60-70% (0,6-0,7). L'effetto somiglierà a quello di una superficie di tavolo perfettamente lucida, nella quale si specchiano gli oggetti, ma nella quale è ancora riconoscibile anche il motivo di righe. Ad ogni modifica del mondo sarà tuttavia necessario tenere presente quali colori sono possibili con la Palette dei colori impiegata in quel momento. Se si vuole per es. aggiungere un oggetto verde e la Palette non contiene il verde, la routine plot( ) sceglierà un altro colore qualunque, che riterrà adeguato. Lo stesso vale per ombreggiature ecc. Nel caso in cui l'immagine a colori che si è prodotta non ci soddisfi, occupiamoci prima di tutto della Palette dei colori. Questo punto riveste un ruolo importante anche se si lavora in HAM.

Ciò che manca ancora in questo programma è una funzione che traduca in strutture di programma un file di definizioni in ASCII. Attualmente, per ogni modifica del mondo, è necessario ricompilare e linkare. Nell'altro caso invece sarà necessario apportare una sola modifica al file ASCII, tramite il buon vecchio ED, e trasferire il file al programma.

Sarebbe pensabile anche un piccolo editor grafico, tramite il quale posizionare sullo schermo gli oggetti come se fossero modelli in filo metallico. Solo dopo potrebbe venire chiamato il programma vero e proprio di Ray-Tracing.

Diversamente potremo aggiungere nuovi oggetti, nuove strutture di superfici, oppure inserire una routine in farbintens( ), che modifichi il vettore della normale secondo determinati criteri. Con ciò potremmo simulare una superficie ruvida o accidentata ecc. Non c'è nessun limite alla nostra fantasia!

Al lavoro quindi, e buon divertimento!

6.5 Corpi trasparenti oppure: legge di rifrazione

Esiste un'altra caratteristica possibile dei corpi, della quale non ci siamo ancora occupati: la trasparenza. Gli oggetti in vetro o plexiglas ecc. possono essere trasparenti. E' possibile vedere cose che si trovano dietro questi oggetti. A seconda del grado di trasparenza e dello spessore dell'oggetto trasparente esse potranno apparire lattiginose oppure nitide, chiare come nell'originale oppure più scure. Se un oggetto è trasparente solo per determinati colori (es. vetro azzurro), gli oggetti posti dietro di esso appariranno solo nella parte di colore che viene lasciata passare (azzurro).

Anche quando una cosa è perfettamente trasparente per la luce al cento per cento, il nostro occhio è in grado di riconoscerla. Ciò dipende dalla cosiddetta rifrazione della luce. Quando un raggio di luce penetra da un mezzo (aria) in un altro mezzo (vetro o acqua ecc.), la sua direzione si modifica. Risparmiamoci di analizzarne la motivazione, che ha a che vedere con la meccanica dei quanti, e prendiamolo come dato di fatto.

A causa di questa rifrazione della luce, può accaderci di vedere, dietro un oggetto trasparente, delle cose in una posizione, mentre si trovano in una posizione completamente diversa. Immergiamo per es. una asticella in un bicchiere di acqua. Ci sembrerà che l'asticella sia leggermente piegata nel punto in cui essa entra nell'acqua. Lo stesso principio viene utilizzato anche per le lenti dei binocoli o dei fotobiettivi.

Quindi la luce modifica la propria direzione quando entra in un oggetto trasparente. Inoltre essa modificherà altrettanto la propria direzione quando ne esce. La legge fisica che descrive questo fenomeno viene chiamata legge di rifrazione ed è la seguente (ved. Fig. 6.10):

$n_1 * \sin(a_1) = n_2 * \sin(a_2)$

n_1, n_2 indici di rifrazione dei due mezzi (costanti del materiale) $n_1, n_2 \geqq 1$
a_1 angolo di incidenza fra la normale $\vec{n}$ e il raggio in entrata $\vec{S}$
a_2 angolo di calcolo tra la normale $\vec{n}$ ed il raggio rifratto $\vec{Sb}$

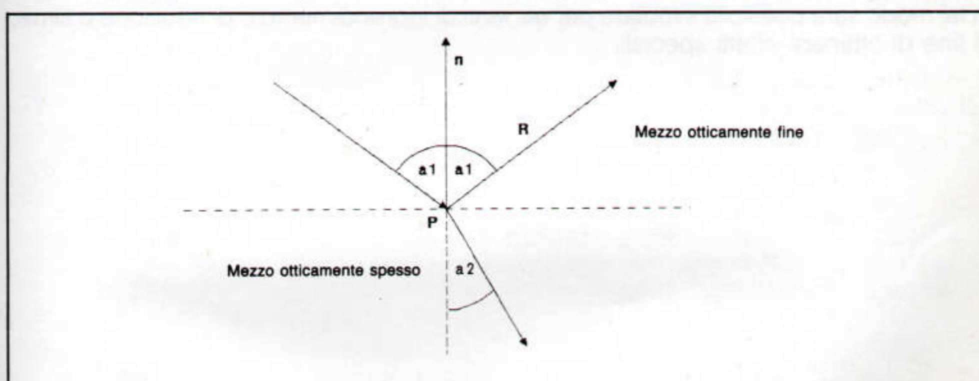


Fig. 6.10 Legge di rifrazione

Se un raggio di luce entra in un mezzo otticamente spesso (es. vetro, indice di rifrazione elevato), proveniendo da un mezzo otticamente sottile (es. aria, dotata del più basso indice di rifrazione, n_1 è circa uguale a 1), il raggio rifratto avrà, nel secondo mezzo, un andamento quasi verticale ($n_1 < n_2 => a_2 < a_1$). Nel caso contrario, al passaggio da un mezzo otticamente spesso ad uno otticamente sottile, esso sarà molto lontano dalla verticale ($n_1 > n_2 => a_2 > a_1$). In quest'ultimo caso può anche aggiungersi il fatto che a2 sia uguale a 90° ($\sin(a_2) = 1$). Dal momento in cui il seno non può essere maggiore di 1, nel caso di a_2 maggiore di 90 si giungerà ad una riflessione totale (specchiatura) del raggio all'indietro nel mezzo più spesso (quindi nessuna rifrazione).

C'è ancora una cosa da considerare: non tutta la luce che arriva al confine tra due mezzi entra nel secondo mezzo (e viene quindi rifratta). Una parte di essa, infatti, viene riflessa normalmente, come abbiamo già visto per i corpi normali. Maggiore è l'angolo di incidenza a1, maggiore sarà anche il quantitativo di luce che viene riflesso. Con

$a_1 = 0$ (ingresso verticale) anche la riflessione è uguale a 0. E' possibile osservare questo effetto anche nei nostri oggetti di uso quotidiano: in un vetro di finestra si specchia quasi tutto, per cui è solo a fatica che riusciamo a guardare dentro. Tuttavia, quanto più perpendicolarmente guardiamo verso il vetro, tanto minori diverranno le specchiature di disturbo.

Un corpo trasparente presenta, come ogni altro corpo, anche delle riflessioni a specchio e diffuse. La dimensione della riflessione dipenderà quindi dall'angolo di incidenza della luce.

Possiamo ampliare il nostro modello di illuminazione, aggiungendo ad esso questa legge di rifrazione della luce in caso di corpi trasparenti. Il raggio dello sguardo dovrà venire deviato secondo le regole della legge di rifrazione (tenendo conto eventualmente anche della riflessione e della riflessione totale sia per il raggio della vista che per eventuali fonti di luce) per penetrare nel corpo. Per la sua uscita dovremo tenere conto delle stesse regole. A questo punto, il raggio ripetutamente deviato a seconda delle situazioni, potrà venire tracciato normalmente come un normalissimo raggio luminoso. In tal modo sarà possibile simulare per es. lenti di ingrandimento o di riduzione o simili, al fine di ottenere effetti speciali.

CAPITOLO 7

Un'altra applicazione:
Solidi di rotazione

Questo capitolo si occupa di una semplice ma molto interessante applicazione della grafica tridimensionale. Conosciamo già alcuni aspetti dai capitoli precedenti, e nel presente troveremo ulteriori suggerimenti ed ampliamenti.

Ci occuperemo dei cosiddetti solidi di rotazione. Si tratta parzialmente di immagini complicate, tuttavia relativamente semplici da costruire e di conseguenza applicabili in molti sistemi tridimensionali.

7.1 Cosa sono i solidi di rotazione?

Finora ci siamo occupati della rappresentazione tridimensionale di corpi relativamente semplici. Ciò dipendeva principalmente dal fatto che l'input di molti punti angolari e di molte linee è piuttosto laborioso. Di conseguenza, le immagini rappresentate erano spesso piatte e spigolose, in quanto l'implementazione di corpi curvi (a prescindere dalla tecnica di Ray-Tracing) crea numerosi problemi in relazione al numero di angoli e lati.

Qui invece introdurremo una tecnica che rende possibile in maniera molto più semplice la produzione di tali corpi rotondi. Limitiamoci tuttavia ai cosiddetti solidi di rotazione simmetrici sull'asse, cosa che non rappresenta una limitazione particolare, in quanto potremo combinare tali corpi rotondi con qualunque corpo angolare.

Il trucco sta nel fatto che verranno immessi solo pochi punti ben determinati, mentre il resto verrà calcolato dal programma sulla base di una regola di calcolo per i solidi di rotazione.

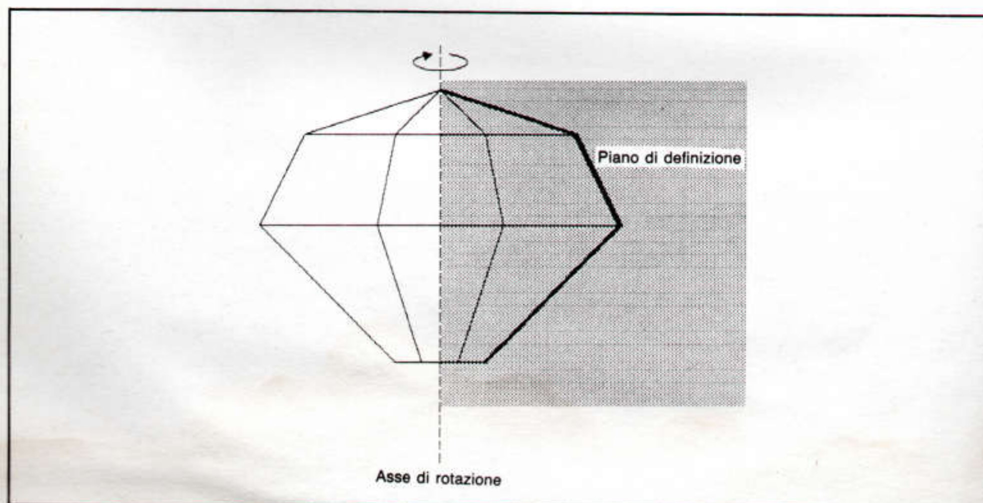


Fig. 7.1 Generazione di un solido di rotazione

L'input dei punti necessari ha luogo in un sistema di coordinate bidimensionali, cioè sul piano x-y con $z=0$. In tal modo è possibile creare un'immagine a piacere con un numero di punti a piacere riprodotta ripetutamente. Ogni punto è collegato con il suo precedente da una linea (vedi Fig. 7.1). Nel caso più semplice, collegheremo semplicente due punti l'uno all'altro.

Al fine di superare la bidimensionalità del piano, lo ruoteremo attorno all'asse y di un angolo a piacere, non troppo grande né troppo piccolo, per cui ogni singolo punto e ogni singola linea del piano verranno ruotati. Le nuovi posizioni dei punti vengono annotate, come accadeva per le vecchie posizioni, in un elenco dei punti. Anche le linee devono venire memorizzate. Il passo successivo è quello di ruotare un'altra volta quest'ultimo piano dello stesso angolo. I punti e le linee derivanti vengono di nuovo memorizzati ecc. Questa operazione viene ripetuta varie volte, finché non viene completata una rotazione completa di 360°. Il risultato ha l'aspetto di un corpo rotondo, approssimato dalle molte rotazioni.

Ora, se ogni punto ruotato viene collegato con il proprio punto originale, il corpo sarà composto da molti piccoli poligoni rettangolari. Quanto più saranno piccoli tali poligoni, tanto più rotondo apparirà l'oggetto.

Generiamo quindi un elenco di superfici dall'elenco dei punti e delle linee. Tutti gli elenchi vengono collegati fra loro in un oggetto, che noi potremo manipolare esattamente come un oggetto tridimensionale normale, come abbiamo già visto: l'oggetto potrà a sua volta essere in scala, spostato, ruotato o proiettato. Superfici potranno coprire altre superfici ecc.

L'unico problema è quello di riservare uno spazio di memoria sufficiente, dal momento che il numero dei punti, linee e superfici viene continuamente sommato. Se w è l'angolo per la produzione di un solido di rotazione da un piano, e P è il numero dei punti nel piano, per il numero dei punti, linee e superfici da memorizzare varrà:

Punti: Pges = 360/w * P
Linee: Lges = 360/w * (2*P-1)
Superfici: Fges = 360/w * (P-1)

w dovrebbe essere sempre un divisore di 360°, diversamente il quoziente 360/w dovrà venire arrotondato. Con un angolo di rotazione di 10° con solo 6 punti di definizione su un piano, avremo:

Numero dei punti: Pges = 216
Numero delle linee: Lges = 396
Numero delle superfici: Fges = 180

7.2 Applicazione per grafici tridimensionali

Il seguente programma in Basic produce un solido di rotazione che dovremo editare, e ce lo presenta completamente trasformato, proiettato con superfici nascoste sullo schermo. In esso inoltre si tiene conto anche di una fonte di luce, la cui posizione è selezionabile a piacere. Vediamolo subito:

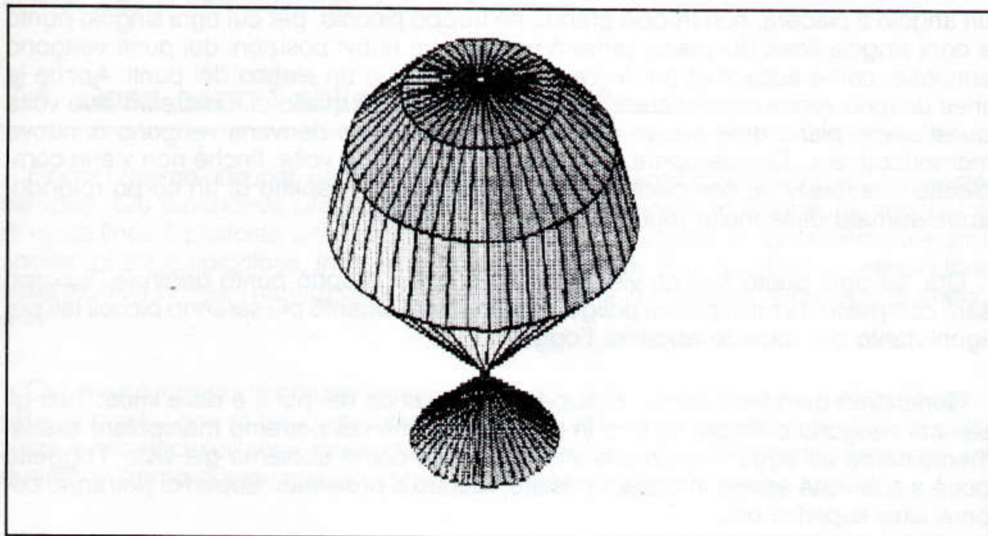


Fig. 7.2 Solido di rotazione 1

```
'**************************
'**                      **
'** Solidi di Rotazione  **
'**                      **
'**************************

'Calcolo fabbisogno memoria

'Impostazione valori massimi a'seconda della memoria disponibile
DATA 15, 45

READ MAX.ECKP%     'N.ro max. dei plunti angolari
READ MAX.ROTS%     'N.ro max. delle rotazioni

'Fabbisogno memoria per matrici superfici:
max.flaech& = MAX.ROTS% * (MAX.ECKP%-1) * 26
'Fabbisogno memoria per matrici punti:
max.punkte& = MAX.ROTS% * MAX.ECKP% * 24
```

```
'Fabbisogno totale di memoria:
speicher& = max.flaech& + max.punkte& + 30000

IF FRE(-1)+FRE(0) <= speicher& + 64000& THEN
  PRINT "Spazio mem. insufficiente!!!"
  PRINT : PRINT "Spazio  necessario  ora:"
  PRINT "per  Punti:  ";max.punkte%
  PRINT "per Superf.: ";max.flaech%
  PRINT "totale:       ";speicher% : PRINT
  PRINT "Delimitare  nel  programma  i  valori  massimi"
  PRINT "per il numero dei punti angolari e/o"
  PRINT "il numero delle rotazioni"

  PRINT : PRINT "Valori Massimi attuali per:"
  PRINT "Punti ang.:  ";MAX.ECKP%
  PRINT "Rotazioni: ";MAX.ROTS%
  END
END IF

' Riserva memoria:
CLEAR ,speicher&
RESTORE

READ MAX.ECKP%   'Numero max. di punti angolari
READ MAX.ROTS%   'Numero max. di rotazioni

PI = 3.141593

'Apertura nuovo schermo con finestra
anz.farben% = 16
SCREEN 2,640,200,4,2
wx% = 631
wy% = 186
WINDOW 2,"Solidi di rotazione",(0,0)-(wx%,wy%),4+8,2

blauf = 0       'Fattore blu
gruenf = 0      'Fattore verde
rotf = 1        'Fattore rosso
blauadd = .1    'Aggiunta di blu
gruenadd = .1   'Aggiunta di verde
rotadd = 0      'Aggiunta di rosso

'Assegnazione colori:
FOR i=2 TO anz.farben%-1
  farbe = INT(i*100/15 + .5)/100
  rot   = farbe*rotf+rotadd
  gruen = farbe*gruenf+gruenadd
  blau  = farbe*blauf+blauadd
  IF rot>1 THEN rot=1
  IF gruen>1 THEN gruen=1
  IF blau>1 THEN blau=1

  PALETTE i,rot,gruen,blau
NEXT i
```

```
PALETTE 0,0,0,0         'Sfondo nero
PALETTE 1,1,.8,.13      'Colore cornice

' Parametri di trasfromazione di Start
' Scala
sx = 1 : sy = 1 : sz = 1

' Traslazione
tx = 0 : ty = 0 : tz = 0

' Rotazione
rx = 10 : ry = 0 : rz = 0

' Osservatore
bx = 0 : by = 0 : bz = 5000

' Fonte di luce
lq.x& = 900 : lq.y& = 1000 : lq.z& = -500

' Intensita'' luce/di superficie
li.int = .7 : fl.int = 1 : hin.int = .3

' Traslazione di un piano
tex = INT(wx%/2) : tey = INT(wy%/2)

' Messa in scala di un piano
sex = 1 : sey = 1

' Numero di Start delle rotazioni per solido
anz.rots% = 8

' Trasformazione angolo rotazione in RAD:
rx = rx*PI/180 : ry = ry*PI/180 : rz = rz*PI/180

'Valore incremento rotazione (Gradi):
dig = 10

di = PI * dig/180                 'Trasformazione in RAD

'Incremento osservatore:
binc = 100

POKE WINDOW(8)+27,1               'Colore dell Area-Outline-Pen
flags = WINDOW(8)+32
POKEW flags,PEEKW(flags) OR 8 'Impostaz. dell'Area-Outline-Flag

'Determinazione solido di rotazione, preparazione
'in matrici dei dati oggetto e
'definizione delle seguenti matrici:
'xr(), yr(), zr()      - Coordinate dei punti dello spazio
'xe(),ye(),ze()        - Coordinate trasformate
'fld%(,)               - Definizioni superfici
'flz%(,)               - superfici da tracciare
'sort.f%()             - Indici selezionati delle superfici
```

```
'mit.z()               - Valori z medi di matrice delle superf.
'farben()              - Intensita' colori di tutte le superf.
'                        da tracciare.

get.rotkoerper(-1)


'Loop principale:
'****************

flag%=0
WHILE flag%<>1
  IF flag%<>-999 THEN

    transform         'Trasformazione di tutti i punti
    verdecke          'Filtraggio superfici nascoste
                      '(e determinazione colori)
    projektion        'Proiettare tutti i punti

  END IF

  CLS                 'Cancellazione finestra

  PATTERN &HFFFF      'Motivo di linee = a zig zag

  'Disegno oggetto
  FOR i=0 TO afz%-1                       'Tutte le superfici
    flaech.nr% = sort.f%(i)               'Nr. superfici da tracciare
    FOR k=0 TO 3                          'Tutti i punti angolari della superf.
      punkt.nr% = flz%(flaech.nr%,k)      'Numero punto
      x% = tex + sex*xe(punkt.nr%)        'Coordinate del punto
      y% = tey - sey*ye(punkt.nr%)

      IF x%<0     THEN x% = 0
      IF x%>=wx% THEN x% = wx%-1
      IF y%<0     THEN y% = 0
      IF y%>=wy% THEN y% = wy%-1

      AREA (x%,y%)

    NEXT k

    COLOR INT(farbe(flaech.nr%)*14)+2 'Impostazione valore colore
    AREAFILL 0                        'Tracciatura superficie
  NEXT i

  maus% = 0 : flag%=0
  WHILE maus%<>1 AND flag%=0
    SLEEP             'Attesa evento
    '
    'Assunzione delle coordinate del Mouse
    'come nuove coordinate del punto zero
    maus%=MOUSE(0)
```

```
IF maus%=1 THEN
  tex = MOUSE(3)
  tey = MOUSE(4)
  flag% = -999  'Flag per solo disegno
END IF
ch% = ASC(INKEY$+CHR$(0))
IF ch%=31 THEN  'Cursore a sinistra  =>
  rz=rz-di      'diminuzione angolo rotazione asse z
  flag% = -1    'Flag per nuovo disegno
END IF
IF ch%=30 THEN  'Cursore a destra =>
  rz=rz+di      'incremento angolo rotazione asse z
  flag% = -1    'Flag per nuovo disegno
END IF
IF ch%=28 THEN  'Cursore in alto =>
  rx=rx-di      'diminuzione angolo rotazione asse x
  flag% = -1    'Flag per nuovo disegno
END IF
IF ch%=29 THEN  'Cursore in basso =>
  rx=rx+di     .'incremento angolo rotazione asse x
  flag% = -1    'Flag per nuovo disegno
END IF
IF ch%=127 THEN 'Del => limitazione superfici si/no
  POKEW flags,PEEKW(flags) XOR 8
  flag% = -999  'Flag per solo disegno
END IF
IF ch%=43 THEN  '+ => scala verso l'alto
  sx = sx+.1
  sy = sy+.1
  sz = sz+.1
  flag% = -1
END IF
IF ch%=45 THEN  '- => scala verso il basso
  sx = sx-.1
  sy = sy-.1
  sz = sz-.1
  flag% = -1
END IF
IF ch%=56 THEN  '8 => allontanamento osservatore
  bz = bz-binc
  flag% = -1
END IF
IF ch%=50 THEN  '2 => avvicinamento osservatore
  bz = bz+binc
  flag% = -1
END IF
IF ch%=139 THEN 'Help => Rotazione = 0
  rx = 0
  ry = 0
  rz = 0
  flag% = -1
END IF
IF ch%=27 THEN  'Esc => nuova determinazione dell'oggetto
  get.rotkoerper(0)
  flag% = -1
```

```
END IF
IF ch%=129 THEN 'F1 => modifica numero rotazioni
  LOCATE 1,1 : PRINT "Nuovo input numero di rotazioni:"
  PRINT "Numero attuale di rotazioni: ";anz.rots%
  PRINT "Nuovo numero di rotazioni:      ";: INPUT anz.rots%
  IF anz.rots% > MAX.ROTS% THEN
    anz.rots% = MAX.ROTS%
    PRINT
    PRINT "Numero troppo alto! E' stato scelto il"
    PRINT "valore massimo permesso ";MAX.ROTS%;""
  END IF
    get.rotkoerper(1)
    flag% = -1
  END IF
  IF ch%=8 THEN    'Tasto indietro => cambio colore
    'Rotazione parametri colore:
    zwis     = rotf
    rotf     = gruenf
    gruenf   = blauf
    blauf    = zwis
    zwis     = rotadd
    rotadd   = gruenadd
    gruenadd = blauadd
    blauadd  = zwis

    'Attribuzione colori:
    FOR i=2 TO anz.farben%-1
      farbe = INT(i*100/15 + .5)/100
      rot   = farbe*rotf+rotadd
      gruen = farbe*gruenf+gruenadd
      blau  = farbe*blauf+blauadd
      IF rot>1 THEN rot=1
      IF gruen>1 THEN gruen=1
      IF blau>1 THEN blau=1

      PALETTE i,rot,gruen,blau
    NEXT i
  END IF

  'Se finestra chiusa -> Fine
  IF WINDOW(7)=0 THEN
    flag%=1
  END IF

 WEND
WEND

WINDOW OUTPUT 1
SCREEN CLOSE 2

END
```

```
'Calcolo del solido di rotazione:

SUB get.rotkoerper(flag%) STATIC
  ' flag% = -1 : Prima chiamata
  ' flag% =  0 : Input completamente nuovo del solido
  ' flag% =  1 : Ora ricalcolare il solido

  SHARED anz.rots%, MAX.ROTS%, MAX.ECKP%
  SHARED wx%, wy%, PI

  IF flag% = 0 OR flag% = -1 THEN
    GOSUB edit.koerper            'Editaggio solido
  END IF

PRINT : PRINT "Attendere, sto calcolando"

' Dimensionamenti e dichiarazioni:

IF flag%<>-1 THEN
  ERASE xr, yr, zr, xe, ye, ze
  ERASE fld%, flz%, sort.f%, mit.z, farbe
END IF

'Coordinate dei punti:
SHARED ap%

ap% = anz.rots% * ecken%        'Numero di tutti i punti

DIM SHARED xr(ap%-1),yr(ap%-1),zr(ap%-1)
DIM SHARED xe(ap%-1),ye(ap%-1),ze(ap%-1)

'Definizioni delle superfici:
SHARED afd%, afz%

afd% = anz.rots% * (ecken%-1) 'Numero di tutte le superfici
afz% = afd%           'Numero della superfici da disegnare

DIM SHARED fld%(afd%-1,3),flz%(afz%-1,3)
DIM SHARED sort.f%(afz%-1),mit.z(afz%-1)
DIM SHARED farbe(afz%-1)


'Determinazione di tutti gli altri punti e superf.

FOR i=0 TO ecken%-1
  xr(i) = xrot%(i)-INT(wx%/2) 'Trasferimento coordinate angolo
  yr(i) = INT(wy%/2)-yrot%(i) 'nella matrice del punto spaziale
  zr(i) = 0
NEXT i

ap.% = ecken% : alt% = 0
afd.% = 0
inc.w = 2*PI/anz.rots%          'Calcolo incremento angolare
```

```
'Calcolo del solido di rotazione:
FOR w=inc.w TO 2*PI+inc.w/5 STEP inc.w
  IF ap.% = ap% THEN
    ap.% = 0                    'Punti iniziali = punti finali
  ELSE
    si = SIN(w) : co = COS(w)

    'Rotazione del piano di definizione:
    FOR i=0 TO ecken%-1
      x = xr(i)                 'Coordinate prima riga
      z = zr(i)
      xr(ap.%+i) = x*co + z*si'ruotare attorno all'asse y
      yr(ap.%+i) = yr(i)
      zr(ap.%+i) = -x*si + z*co
    NEXT i
  END IF

  'Costruzione superfici:
  FOR i=0 TO ecken%-2
    fld%(afd.%+i,0) = alt%+i   'Memorizzazione numeri punti
    fld%(afd.%+i,1) = alt%+i+1
    fld%(afd.%+i,2) = ap.%+i+1
    fld%(afd.%+i,3) = ap.%+i
  NEXT i

  alt% = ap.%
  ap.% = ap.% + ecken%
  afd.% = afd.% + ecken%-1

  NEXT w

  EXIT SUB

  ' Editaggio contorni del solido di rotazione
  edit.koerper:
    IF flag%<>-1 THEN
      ERASE xrot%, yrot%
    END IF
    DIM xrot%(MAX.ECKP%-1), yrot%(MAX.ECKP%-1)

    ecken% = -1

    GOSUB zeichne

    WHILE (ch%<>27 OR ecken%<=0)     'Fine con ESC
      SLEEP                          'Attesa evento

      maus% = MOUSE(0)
      IF maus%=1 THEN                'Tasto Mouse premuto

        ecken% = ecken%+1
        IF ecken% >= MAX.ECKP% GOTO rotraus

        xrot%(ecken%) = MOUSE(3)     'Annotazione coordinate Mouse
        yrot%(ecken%) = MOUSE(4)     'come punto angolare di rotazione
```

```
      IF ecken%=0 THEN
        PSET (xrot%(0), yrot%(0)) 'Posizionamento cursore grafico
      ELSE                        'oppure: tracciatura di riga
        LINE -(xrot%(ecken%), yrot%(ecken%))
      END IF
    END IF

    ch% = ASC(INKEY$+CHR$(0))      'Prelevamento tasto
    IF ch%=127 AND ecken%>=0 THEN 'Del=>cancellazione ultimo punto
      ecken% = ecken%-1
      GOSUB zeichne
      IF ecken% > 0 THEN
        FOR i=0 TO ecken%-1        'nuovo disegno:
          LINE (xrot%(i), yrot%(i))-(xrot%(i+1),yrot%(i+1))
        NEXT i
      ELSE
        IF ecken% = 0 THEN PSET (xrot%(0), yrot%(0))
      END IF
    END IF
    IF WINDOW(7)=0 THEN            'Chiusura finestra
      WINDOW OUTPUT 1
      SCREEN CLOSE 2
      END
    END IF
  WEND

  rotraus:
  ecken% = ecken% + 1
 RETURN

 zeichne:
  CLS

  COLOR 10
  LOCATE 1,22 : PRINT "Input contorni di un solido di rotazione"

  'Tracciatura del sìstema di coordinate:
  PATTERN &HF0F0
  COLOR 15
  LINE (0,wy%/2)-(wx%,wy%/2)       'Asse di rotazione
  LINE (wx%/2,0)-(wx%/2,wy%)

  PATTERN &HFFFF
  COLOR 1
 RETURN

END SUB

'Trasformazione di tutti i punti nello spazio:
SUB transform STATIC
  SHARED ap%
  SHARED sx,sy,sz,tx,ty,tz,rx,ry,rz
```

```
'Calcolo delle costanti per la rotazione:
si.x = SIN(rx) : co.x = COS(rx)
si.y = SIN(ry) : co.y = COS(ry)
si.z = SIN(rz) : co.z = COS(rz)

A = co.y * co.z
B = co.y * si.z
C = -si.y
D = si.x*si.y*co.z - co.x*si.z
E = si.x*si.y*si.z + co.x*co.z
F = si.x*co.y
G = co.x*si.y*co.z + si.x*si.z
H = co.x*si.y*si.z - si.x*co.z
J = co.x*co.y

FOR i=0 TO ap%-1
  ' Trasformazioni:
  xt = sx*xr(i) + tx            'Scala
  yt = sy*yr(i) + ty            'e traslazione
  zt = sz*zr(i) + tz

  xe(i) = xt*A + yt*B + zt*C    'Rotazioni
  ye(i) = xt*D + yt*E + zt*F
  ze(i) = xt*G + yt*H + zt*J
NEXT i

END SUB

'Proiezioni di tutte le coordinate dello spazio nel piano:
SUB projektion STATIC
  SHARED ap%
  SHARED bx,by,bz

  FOR i=0 TO ap%-1
      ' Proiezione centrale:
      zwis = ze(i) - bz
      xe(i) = bx - bz * (xe(i)-bx)/zwis
      ye(i) = by - bz * (ye(i)-by)/zwis
  NEXT i
END SUB

'Filtraggio di itutte le superfici coperte
'ed ordinamento dei valori z medi:

SUB verdecke STATIC
  SHARED afd%, afz%
  SHARED bx,by,bz

  afz% = 0                'Numero delle superfici da tracciare

  FOR i=0 TO afd%-1
```

```
'Fase 1: analisi del retro della superficie:
'*****************************************

'Determinazione di due vettori indipendenti
'linearmente della superficie:
'v = P2-P1 // w = P3-P1:
'con: P1,P2,P3 = Punti angolari della superficie
P1% = fld%(i,0)
P2% = fld%(i,1)
P3% = fld%(i,2)
P1.x = xe(P1%)
P1.y = ye(P1%)
P1.z = ze(P1%)
v.x = xe(P2%) - P1.x
v.y = ye(P2%) - P1.y
v.z = ze(P2%) - P1.z
w.x = xe(P3%) - P1.x
w.y = ye(P3%) - P1.y
w.z = ze(P3%) - P1.z

'Determinazione del prodotto vettoriale p = v x w:
p.x = v.y*w.z - v.z*w.y
p.y = v.z*w.x - v.x*w.z
p.z = v.x*w.y - v.y*w.x

'Vettore dello sguardo s dall'osservatore a P1:
s.x = P1.x - bx
s.y = P1.y - by
s.z = P1.z - bz

'Calcolo del prodotto scalare q = p * s:
q = p.x*s.x + p.y*s.y + p.z*s.z

'Controllo del segno di q:
IF q>0 THEN

'Identificazione superficie come visibile:
sum.z = 0
FOR k=0 TO 3
  flz%(afz%,k) = fld%(i,k) 'Trasferimento defin. superf.
  sum.z = ze(fld%(i,k)) + sum.z 'formazione della somma z
NEXT k

GOSUB farbe           'Calcolo valore colori delle superf.

'Fase 2: ordinamento valori medi z
'*********************************

'Memorizzazione nella matrice z del valor medio z
mittel  = sum.z / 4
mit.z(afz%) = mittel

'Ordinam. in matrice sort indice di superf. secondo il valor medio z
k = 0
```

```
    'Ricerca punto di ordinamento:
    WHILE (mittel <= mit.z(sort.f%(k)) AND k < afz%)
      k=k+1
    WEND
    IF k <= afz% THEN
      'Spostamento a partire dal punto di ordinamento:
      FOR m=afz% TO k STEP -1
        sort.f%(m+1) = sort.f%(m)
      NEXT m
    END IF
    sort.f%(k) = afz%   'Annotazione Indice x della superficie

    afz% = afz%+1        'Aumento numero superfici visibili

  END IF

 NEXT i                   'Prossima superficie

 EXIT SUB

 'Calcolo intensita' colore superficie:
 farbe:
   SHARED lq.x&, lq.y&, lq.z&
   SHARED li.int, fl.int, hin.int

   ' Vettore dal punto della superf. alla fonte di luce:
   L.x = P1.x - lq.x&
   L.y = P1.y - lq.y&
   L.z = P1.z - lq.z&

   ' Intensita' della luce incidente:
   cos.a = (p.x*p.x + p.y*p.y + p.z*p.z)
   cos.a = cos.a * (L.x*L.x + L.y*L.y + L.z*L.z)
   cos.a = (p.x*L.x + p.y*L.y + p.z*L.z)/SQR(cos.a)

   farb = hin.int*fl.int          'Intensita' sfondo
   IF cos.a < 0 THEN              'Superf. rivolta alla luce?
     farb = farb - li.int*fl.int * cos.a ' SI! determinazione
   END IF                                 ' incidenza luce
   farbe(afz%) = farb - INT(farb) 'solo fra 0 e 1!
 RETURN

END SUB
```

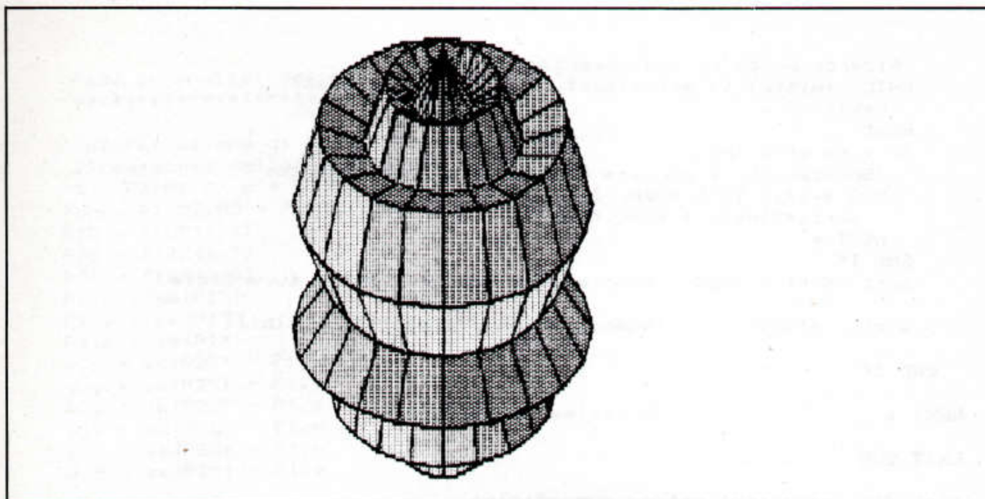


Fig. 7.3 Solido di rotazione 2

Mettiamoci al lavoro: si tratta di nuovo di un programma con il quale potremo lavorare in maniera veramente creativa. Infatti non ci troviamo ad essere solo spettatori, bensì viviamo questa esperienza insieme con il nostro computer. Quando lanciamo il programma appare uno schermo quasi vuoto, nel quale è presente solo un sistema di coordinate bidimensionale con assi x ed y ed una scritta: "Input dei contorni di un solido di rotazione". Questo è il punto nel quale potremo definire tramite il mouse i piani di contorno del solido di rotazione. La linea verticale marca contemporaneamente la posizione dell'asse di rotazione, attorno al quale tutti i punti da noi immessi verranno in seguito ruotati passo passo, al fine di produrre il solido di rotazione (vedi sopra).

Portiamoci quindi con il mouse su di un punto a piacere dello schermo (possibilmente non troppo lontano dall'asse di rotazione) e premiamo il pulsante sinistro del mouse. Apparirà un punto. Spostiamo il mouse a piacere e premiamo di nuovo il pulsante del mouse. Il programma traccerà immediatamente una linea dal primo al secondo punto. Continuiamo in questa maniera finché non avremo marcato e collegato 5 o 6 punti. In tal modo abbiamo determinato il contorno del nostro solido di rotazione. Se questo contorno non ci piace, premiamo semplicemente il tasto <Del>, e l'ultimo punto (e la linea ad esso collegata) spariranno. Possiamo modificare questo contorno ripetutamente e, quando siamo pronti, premiamo il tasto Esc per partire. Riassumiamo brevemente ancora i comandi:

<Tasto sinistro del mouse>	Definizione di un punto angolare
<Del>	Cancellazione di un punto angolare
<Esc>	Calcolo e rappresentazione del solido di rotazione
<Window close>	Fine del programma

Dopo alcuni istanti apparirà sullo schermo il risultato. Se abbiamo messo molti punti

angolari, è probabile che sia necessario qualche minuto di tempo di calcolo. Se possibile, è opportuno posizionare il primo e l'ultimo punto il più vicino possibile all'asse di rotazione, se non direttamente su di esso.

A questo punto abbiamo a disposizione una serie di comandi, al fine di modificare l'apparizione del nostro solido di rotazione:

<Tasto mouse sin.>	Spostamento del solido di rotazione
<cursore sin.>	Rotazione a sinistra attorno all'asse z
<cursore ds.>	Rotazione a destra attorno all'asse z
<cursore verso l'alto>	Rotazione a sinistra attorno all'asse x
<cursore verso il basso>	Rotazione a destra attorno all'asse x
<Del>	Attivazione/disattivazione limitazione superifici
<+>	Ingrandimento dell'oggetto
<->	Riduzione dell'oggetto
<8>	Allontanare l'osservatore
<2>	Avvicinare l'osservatore
<Help>	Impostare a 0 tutti gli angoli di rotazione
<Spazio indietro>	Cambio colori
<F1>	Modifica del numero di rotazioni
<Esc>	Sviluppo di un nuovo solido di rotazione
<Window close>	Fine programma

Abbiamo quindi a disposizione una scelta molto vasta. Con F1 modificheremo il numero di rotazioni, senza tuttavia perdere i contorni del corpo. Dopo F1 immetteremo semplicemente il numero delle rotazioni desiderate, ed il programma ricalcolerà completamente il solido di rotazione. La posizione della fonte di luce ed altri parametri potranno venire modificati direttamente nel programma.

A seconda del numero di punti angolari e del numero di rotazioni, aumenterà o diminuirà il tempo di calcolo, trasformazione, e proiezione di un nuovo solido di rotazione, nonché il controllo della visibilità delle superfici. Con valori molto alti, il calcolo potrà impiegare anche alcuni minuti. Un programma in C o in Assembler potrebbe essere molto più veloce, e renderebbe possibile anche l'animazione.

Passiamo quindi alla descrizione del programma. Tutto il lavoro di calcolo è un grosso ''mangiamemoria''. Quindi è consigliabile verificare all'inizio del programma se si dispone di memoria sufficiente o se è il caso di riservarla. La dimensione della memoria da riservare dipende quasi esclusivamente dai valori massimi dei numeri di rotazioni e di angoli. Tali numeri si trovano nelle prime righe del programma, in una istruzione DATA (in questo caso 15 e 45). Nel caso in cui la memoria non dovesse bastare, modificheremo taii valori. Se abbiamo a disposizione più memoria, potremo aumentarli senza problemi. Normalmente il programma ci segnalerà quando stiamo per occupare troppa memoria. Potrà comunque verificarsi un caso di ''OUT OF MEMORY'', dovuto al fatto che, anche se è presente una memoria sufficiente, non si tratta di memoria contigua.

Le inizializzazioni che seguono ci sono già note grazie ai capitoli precedenti. Anche le strutture di base in molte routine dovrebbero già esserci note dal capitolo relativo alle linee e superfici nascoste. Sono state apportate solo modifiche minime (principalmente per la precisione di calcolo). Torniamo quindi all'essenziale, cioè alla routine **get.rotkoerper( )**. Essa permette la creazione di un nuovo solido di rotazione e/o calcola tutti i punti e le superfici di questo corpo, a seconda del contenuto del Flag%.

La creazione del corpo ha luogo nel sottoprogramma interno edit.koerper. Qui il programma attende la pressione dei tasti del mouse, annota le coordinate nelle matrici xr( ), yr( ) e zr( ), traccia una linea dall'ultimo punto e ci dà la possibilità di cancellare di tali linee. Niente di difficile, quindi.

Il nucleo è costituito da un grosso loop FOR...NEXT, dove l'angolo w viene incrementato passo passo (siamo noi stessi a determinare il numero dei passi). Nel loop il programma ruota tutti i punti da noi immessi di un angolo w attorno all'asse y, per cui conosce le coordinate dei punti del piano di rotazione successivo. Quindi egli costruisce, sempre con un poligono di 4 punti, le superfici del corpo. Questo è tutto. Le matrici verranno quindi trasferite alle routine responsabili dell'output.

CAPITOLO 8

Appendice

8.1 Elementi matematici di base per la grafica computerizzata

Nelle pagine seguenti troveremo di nuovo un breve riassunto dei principi matematici più importanti, nonché delle norme e dei procedimenti, necessari nel presente libro per la comprensione della grafica computerizzata tridimensionale. Si tratta alcune volte della ripetizione di problematiche già state chiarite nei capitoli corrispondenti. I lettori che hanno avuto delle difficoltà con i problemi matematici di questo libro potranno consultare questa appendice al fine di rinfrescare le nozioni necessarie. Per il calcolo vettoriale, si consiglia tuttavia anche il Capitolo 4.3, mentre per il calcolo matriciale si consiglia il Capitolo 3.2.1.

8.1.1 Funzioni trigonometriche

Le cosiddette funzioni trigonometriche (seno, coseno, tangente ecc.) ci sono forse già note dalla scuola. Proprio per questo cerchiamo di riassumere ancora una volta che cosa si intende con esse e quali regole esistono.

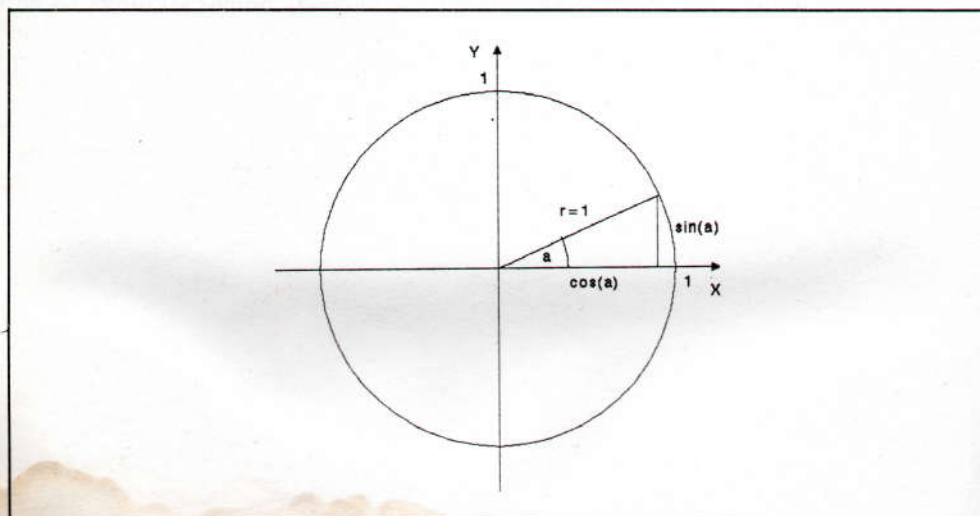


Fig. 8.1 Definizione delle funzioni trigonometriche

Le funzioni trigonometriche vengono definite nel cosiddetto cerchio unitario (ved. Fig. 8.1), cioè un cerchio con il raggio 1 e con il centro nel punto (0,0). Se si traccia una perpendicolare dal raggio c sull'asse x, otterremo un triangolo rettangolo con i lati c, a e b e con l'angolo retto tra i lati a e b. La lunghezza di questi due lati dipende solo dall'angolo w, che è formato dal raggio c e dall'asse x. Le lunghezze a e b vengono definite tramite due nuove funzioni dell'angolo w:

a = sin(w) (che si legge: "seno di w")
b = cos(w) (che si legge: "coseno di w")

Vediamo quindi che maggiore è w, maggiore sarà anche a (cioè sin(w)) e minore diverrà b (cos(w)). Se w è maggiore di 90°, b assumerà valori negativi, a diventerà sempre più piccolo ecc. Dopo 360°, il ciclo ricomincia da capo. La funzione è quindi periodica, con un periodo di 360°. Se si inseriscono queste funzioni in un sistema di coordinate (sull'asse x i valori per w, sull'asse y quelli per sin(w) oppure cos(w)), otterremo la nota funzione sinusoidale ondulata. La funzione del coseno ha lo stesso aspetto, è solo spostata di 90° (ved. Fig. 8.2). Il valore massimo per le funzioni sinusoidali è di 1, il più basso è di -1. La funzione, quindi, oscilla sempre fra questi due valori.

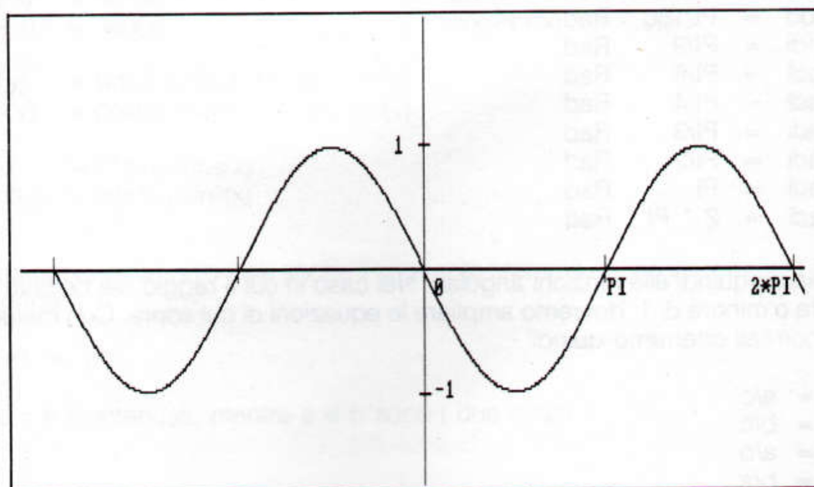


Fig. 8.2 Funzione sinusoidale

Le funzioni angolari terza e quarta, anch'esse utilizzate spesso, vengono definite come segue:

tan(w) = sin(w)/cos(w)
cot(w) = cos(w)/sin(w) = 1/tan(w)

La prima, tan(w), viene chiamata Tangente, la seconda Cotangente. Nel cerchio unitario la tangente è sempre uguale a 1.

Spesso, ma non è il nostro caso, gli angoli vengono indicati in Radianti, che è la lunghezza dell'arco sotteso da un angolo w nel cerchio avente il raggio 1. Un cerchio completo (360°) avrà quindi una lunghezza d'arco di 2*PI. PI è la nota costante che esprime il rapporto tra circonferenwa e diametro:

PI = 3.14159265

Le formule per la trasformazione di un angolo da Gradi (g) in Radianti (r) e viceversa, sono le seguenti:

r = 2*PI*g/360 = PI*g/180
g = 360*r/(2*PI) = 180*r/PI

Ecco alcuni angoli importanti in Gradi e Radianti:

0 Gradi	=	0	Rad
1 Grado	=	PI/180	Rad
10 Gradi	=	PI/9	Rad
30 Gradi	=	PI/6	Rad
45 Gradi	=	PI/4	Rad
60 Gradi	=	PI/3	Rad
90 Gradi	=	PI/2	Rad
180 Gradi	=	PI	Rad
360 Gradi	=	2 * PI	Rad

Ritorniamo quindi alle funzioni angolari. Nel caso in cui il raggio del cerchio diventi maggiore o minore di 1, dovremo ampliare le equazioni di cui sopra. Con metodi geometrici normali otterremo quindi:

sin(w) = a/c
cos(w) = b/c
tan(w) = a/b
cot(w) = b/a

dove c sta per il raggio del cerchio, w per l'angolo tra c e b oppure a e b per i due lati dell'angolo retto del triangolo. A questo punto, indipendentemente dal nostro cerchio, possediamo le formule per il calcolo di un lato di un triangolo rettangolo dati un angolo e un lato.

E' inoltre possibile determinare ciascun angolo di un triangolo rettangolo, noti due lati. c sarà sempre il lato più lungo nel triangolo, cioè quello che si trova di fronte all'angolo retto, chiamato ipotenusa. a e b saranno quindi i due cateti.

Se conosciamo un angolo w in un triangolo rettangolo, il seno di questo angolo costituirà sempre il rapporto del lato che gli sta di fronte rispetto all'ipotenusa, il coseno costituirà il rapporto del lato giacente sull'angolo rispetto all'ipotenusa e la tangente il rapporto del lato frontale rispetto a quello giacente sull'angolo.

Conoscendo la funzione angolare di un angolo, potremo determinare da soli l'angolo stesso tramite le cosiddette funzioni di arco. Infatti vale:

w = arcsin(sin(w))
w = arccos(cos(w))
w = arctan(tan(w))
w = arccot(cot(w))

Si parla infatti di Arcoseno, Arcocoseno, Arcotangente e Arcocotangente (il nostro AmigaBasic conosce solo la funzione di Arcotangente ATN(x)).

Ecco alcune relazioni importanti fra le funzioni angolari:

sin(x)	= cos(PI/2-x)	cos(x)	= sin(PI/2-x)
tan(x)	= cot(PI/2-x)	cot(x)	= tan(PI/2-x)
sin(PI+x)	= -sin(x)	cos(PI+x)	= -cos(x)
tan(PI+x)	= tan(x)	cot(PI+x)	= cot(x)

sin(x+/.y) = sin(x)*cos(y)+/.cos(x)*sin(y)
cos(x+/.y) = cos(x)*cos(y)+/.sin(x)*sin(y)

sin(2*x) = 2*sin(x)cos(x)
cos(2*x) = $\cos^2(x)-\sin^2(x)$

Nel triangolo rettangolo vale anche il seguente teorema, molto importante (Teorema di Pitagora):

$$c^2 = a^2 + b^2$$

dove c è l'ipotenusa, mentre a e b sono i due cateti.

8.1.2 Calcolo vettoriale

Con il termine "vettore" si intende un "tratto orientato" nello spazio o su di un piano (freccia) (normalmente scritto con: $\vec{v}$). Un vettore viene definito tramite il punto di applicazione e il suo estremo. La lunghezza di un vettore $\vec{a}$ viene chiamata anche intensità del vettore e viene indicata con $|\vec{a}|$. Due vettori $\vec{a}$ e $\vec{b}$ sono uguali ($\vec{a} = \vec{b}$), se la loro lunghezza e direzione sono uguali (la posizione nello spazio o sul piano è indifferente). Se un vettore viene spostato parallelamente, si tratterà sempre e comunque di uno e lo stesso vettore.

Se un vettore ha lunghezza 1, lo si chiamerà vettore unitario. Un vettore unitario viene calcolato da $\vec{v}' = \vec{v}/|\vec{v}|$. Il vettore 0 ha lunghezza 0 e la sua direzione è indeterminata.

In alternativa, un vettore $\vec{v}$ potrà venire indicato anche tramite semplici coordinate. In tal caso si ipotizza che il punto di applicazione del vettore si trovi nel punto di origine delle coordinate (possiamo comunque spostarlo a piacere). In questo caso, come coordinate del vettore, verranno indicate solo le coordinate della punta (anche in questo caso sono sufficienti due parametri per il piano e tre per lo spazio):

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

oppure nello spazio:

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Per la lunghezza (grandezza) di un vettore su di un piano, vale:

$$|\vec{v}|^2 = v_x^2 + v_y^2$$

Per i vettori spaziali, avremo:

$$|\vec{v}|^2 = v_x^2 + v_y^2 + v_z^2$$

Con i vettori è possibile anche effettuare dei calcoli. A tal fine valgono tuttavia particolari regole di calcolo:

Moltiplicazione per una cifra:
In questo modo è possibile moltiplicare un vettore con un numero semplice a. Questa moltiplicazione non è da confondere con il prodotto scalare indicato in seguito. Il risultato è anch'esso un vettore. Esso possiede la lunghezza $a^*|v|$. Graficamente si intende con ciò l'allungamento o la riduzione del vettore in questione v. Se a è negativo, verrà invertita la direzione del vettore. In caso di a = 0, il risultato sarà un vettore zero:

$$a^*\vec{v} = a^* \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} a^*v_x \\ a^*v_y \end{pmatrix}$$

nello spazio:

$$a^*\vec{v} = a^* \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} a^*v_x \\ a^*v_y \\ a^*v_z \end{pmatrix}$$

A questo proposito, è importante un altro concetto: l'indipendenza lineare dei vettori. Due vettori sono linearmente indipendenti quando non è possibile trasformare uno nell'altro tramite una moltiplicazione per una costante. In altre parole: quando non giacciono né paralleli né antiparalleli. Di conseguenza verranno detti dipendenti linearmente o colineari due vettori che possiedono la stessa direzione (oppure direzioni opposte) ma non necessariamente la stessa lunghezza.

Addizione/sottrazione di vettori:
E' possibile anche sommare e sottrarre due vettori $\vec{v}$ e $\vec{w}$. Il risultato è anch'esso un vettore. Se vogliamo rappresentare graficamente tale addizione, dovremo semplicemente spostare il punto di applicazione del vettore $\vec{w}$ sulla punta del vettore $\vec{v}$. Il vettore risultante andrà quindi dal piede del vettore $\vec{v}$ alla punta di $\vec{w}$:

$$\vec{v} + \vec{w} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} w_x \\ w_y \end{pmatrix} = \begin{pmatrix} v_x + w_x \\ v_y + w_y \end{pmatrix}$$

Allo stesso modo, nello spazio, avremo:

$$\vec{v} + \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} + \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_x + w_x \\ v_y + w_y \\ v_z + w_z \end{pmatrix}$$

La sottrazione di due vettori ha luogo in maniera analoga:

$$\vec{v} - \vec{w} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} - \begin{pmatrix} w_x \\ w_y \end{pmatrix} = \begin{pmatrix} v_x - w_x \\ v_y - w_y \end{pmatrix}$$

Nello spazio:

$$\vec{v} - \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} - \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_x - w_x \\ v_y - w_y \\ v_z - w_z \end{pmatrix}$$

Regole di calcolo:

$\vec{v}+\vec{w}$	$= \vec{w}+\vec{v}$	$\vec{v}+(\vec{w}+\vec{u})$	$= (\vec{v}+\vec{w})+\vec{u}$
$a^*(b^*\vec{v})$	$= (a^*b)^*\vec{v}$	$(a+b)^*\vec{v}$	$= a^*\vec{v} + b^*\vec{v}$
$a^*(\vec{v}+\vec{w})$	$= a^*\vec{v} + a^*\vec{w}$		
$\|a^*\vec{v}\|$	$= \|a\|^*\|\vec{v}\|$		

Prodotto scalare:

Due vettori $\vec{v}$ e $\vec{w}$ possono anche venire moltiplicati fra loro. Qui però abbiamo due possibilità: per il cosiddetto prodotto scalare $\vec{v}^*\vec{w}$, il risultato sarà un numero semplice (scalare), mentre per il prodotto vettoriale $\vec{v} \times \vec{w}$ otterremo un nuovo vettore.

Il prodotto scalare viene definito come segue:

$$\vec{v}^*\vec{w} = |\vec{v}|^*|\vec{w}|^*\cos(a)$$

In questa formula, a è l'angolo fra i due vettori da moltiplicare. Le grandezze di $\vec{v}$ e $\vec{w}$ sono le lunghezze dei due vettori. In notazione di coordinate, avremo:

$$\vec{v}^*\vec{w} = v_x{}^*w_x + v_y{}^*w_y$$

Mentre, per i vettori spaziali:

$$\vec{v}^*\vec{w} = v_x{}^*w_x + v_y{}^*w_y + v_z{}^*w_z$$

Se i due vettori si trovano perpendicolari varrà:

$\vec{v}^*\vec{w} = 0$ ($\vec{v}$ è perpendicolare a $\vec{w}$)

Regole di calcolo:

$\vec{v}*\vec{w} = \vec{w}*\vec{v}$
$(a*\vec{v})*\vec{w} = a*(\vec{v}*\vec{w})$
$\vec{v}*(\vec{w}+\vec{u}) = \vec{v}*\vec{w} + \vec{v}*\vec{u}$
$\vec{v}*\vec{v} = \vec{v}^2$
$= |\vec{v}|^2$

Attenzione però al fatto che:

$\vec{v}*(\vec{w}*\vec{u}) = (\vec{v}*\vec{w})*\vec{u}$

non vale. In altre parole: dovremo fare attenzione ad eseguire le moltiplicazioni scalari sempre nella sequenza esatta.

Prodotto vettoriale:

Abbiamo già citato il prodotto vettoriale. Si tratta della moltiplicazione di due vettori, che dà come risultato un altro vettore. Questo nuovo vettore $\vec{p} = \vec{v} \times \vec{w}$ ha una lunghezza di:

$|\vec{p}| = |\vec{v} \times \vec{w}| = |\vec{v}|*|\vec{w}|*\sin(a)$

Ciò è d'altra parte anche l'area della superficie del parallelogramma tracciato da $\vec{v}$ e $\vec{w}$. $\vec{p}$ è perpendicolare ai due vettori $\vec{v}$ e $\vec{w}$, cioè penetra nello spazio ($\vec{v}$, $\vec{w}$ e $\vec{p}$ danno come risultato un sistema destrorso). Un vettore perpendicolare viene chiamato anche vettore normale.

Matematicamente, il prodotto vettoriale viene definito come segue:

$$\vec{p} = \vec{v} \times \vec{w} = \begin{pmatrix} v_y*w_z - v_z*w_y \\ v_z*w_x - v_x*w_z \\ v_x*w_y - v_y*w_x \end{pmatrix}$$

Regole di calcolo:

$\vec{v} \times \vec{w} = -(\vec{w} \times \vec{v})$!!!
$(a*\vec{v}) \times \vec{w} = a*(\vec{v} \times \vec{w})$
$\vec{v} \times (\vec{w}+\vec{u}) = \vec{v} \times \vec{w} + \vec{v} \times \vec{u}$
$\vec{v} \times \vec{w}$ = vettore zero (nel caso in cui $\vec{v}$ sia parallelo o antiparallelo rispetto a $\vec{w}$)
$\vec{v} \times \vec{v}$ = vettore zero

Rappresentazione di punti tramite vettori.

Anche un punto può venire interpretato come vettore. In questo caso il vettore andrà dal punto zero al punto stesso:

$$P = \vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{oppure } P = \vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

x, y e z saranno quindi le coordinate del punto. Con P, che significa il punto P, effettueremo i calcoli esattamente come con i vettori normali.

8.1.3 Rette e piani

Rette:

Quanto segue contiene alcune equazioni con le quali è possibile calcolare i punti di una retta:

La più importante è naturalmente l'equazione di una retta su di un piano, che chiunque di noi avrà visto almeno una volta:

$a*x + b*y + c = 0$ (formula implicita)

oppure

$y = m*x + n$ (formula esplicita)

dove m indica l'inclinazione ed n il punto di intersezione con l'asse y, quindi vale:

$$m = \frac{y_2-y_1}{x_2-x_1}$$

Per lo spazio, una retta è rappresentabile come sistema di equazioni lineare (retta di intersezione di due piani):

$a_1*x + b_1*y + c_1*z + d_1 = 0$ e
$a_2*x + b_2*y + c_2*z + d_2 = 0$

E' possibile anche la seguente formula:

$y = m*(x-x_1) + y_1$

dove:

x_1,y_1 un punto della retta
m inclinazione (ved. sopra)

Da ciò deriva la seguente formula:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$ (equazione di due punti)

Un'ultima formula, non molto comune, è:

$$\frac{x}{a} + \frac{y}{b} = 1$$ (formula di sezione dell'asse)

Questa retta interseca l'asse x nel punto P(a,0) e l'asse y nel punto Q(0,b).

L'equazione vettoriale di una retta (per il piano e lo spazio) è:

$$g: \vec{p} = \vec{p}_0 + t^*\vec{a}$$

e, sotto forma di parametri:

$$g: p_x = p_{0x} + t^*a_x$$
$$p_y = p_{0y} + t^*a_y$$
$$p_z = p_{0z} + t^*a_z$$

oppure, come sistema di equazioni:

$$\frac{x-p_{0x}}{a_x} = \frac{y-p_{0y}}{a_y} = \frac{z-p_{0z}}{a_z}$$

dove:

$\vec{p}$ vettore dall'origine delle coordinate al punto P della retta da calcolarsi (oppure più semplicemente: punto P)
$\vec{p}_0$ vettore dall'origine delle coordinate ad un punto P_0 a piacere della retta (oppure, più semplicemente: P_0)
$\vec{a}$ "vettore di direzione", cioè un vettore $\vec{a}$ piacere parallelo alla retta
t fattore di "allungamento" (o riduzione) del vettore a, al fine di raggiungere il punto P

Al fine di determinare una retta ben precisa, avremo bisogno semplicemente dei parametri $\vec{p}_0$ e $\vec{a}$. Da valori a piacere per t determineremo quindi tutti i punti P (oppure vettori $\vec{p}$) della retta.

Piani:
Quanto segue contiene alcune equazioni con le quali è possibile calcolare i punti di un piano nello spazio:

Ecco l'equazione di un piano generale:

$$a^*x + b^*y + c^*z + d = 0$$

La formula di sezione dell'asse è:

$$\frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$$

Questo piano interseca l'asse x nel punto P(a,0,0), l'asse y nel punto Q(0,b,0) e l'asse z nel punto R(0,0,c).

Equazione vettoriale di un piano:

$$e: \vec{p} = \vec{p}_0 + s*\vec{a} + t*\vec{b}$$

oppure, sotto forma di parametri:

$$e: p_x = p_{0x} + s*a_x + t*b_x$$
$$p_y = p_{0y} + s*a_y + t*b_y$$
$$p_z = p_{0z} + s*a_z + t*b_z$$

dove:

$\vec{p}$ vettore dall'origine delle coordinate al punto P del piano da calcolarsi (oppure, più semplicemente: punto P)
$\vec{p}_0$ vettore dall'origine delle coordinate ad un punto P_0 a piacere del piano (oppure, più semplicemente: punto P_0)
$\vec{a},\vec{b}$ vettori a piacere, indipendenti linearmente, sul piano
s,t fattori di "allungamento" (oppure riduzione) dei vettori $\vec{a}$ e $\vec{b}$, al fine di raggiungere il punto P

Al fine di tracciare un piano ben determinato, abbiamo bisogno semplicemente dei parametri $\vec{p}_0$, $\vec{a}$ e $\vec{b}$. Da valori a piacere per s e t calcoleremo quindi tutti i punti P (oppure vettori $\vec{p}$) del piano.

Un'altra forma dell'equazione vettoriale di un piano è la seguente:

$$\vec{n} * \vec{p} + d = 0$$

oppure:

$$(\vec{p}-\vec{p}_0) * \vec{n} = 0$$

dove:

$\vec{n}$ un vettore perpendicolare al piano (di solito con lunghezza 1)
$\vec{p}$ vettore dall'origine delle coordinate al punto P del piano da calcolarsi (oppure, più semplicemente: punto P)
$\vec{p}_0$ vettore dall'origine delle coordinate ad un punto P_0 del piano a piacere (oppure, più semplicemente: punto P_0)
d quota (con d=0 il piano attraversa l'origine)

Per la trasformazione di una delle due equazioni vettoriali di un piano nell'altra, consultare il Capitolo 4.3.

8.1.4 Matrici

Con il termine matrici intendiamo semplicemente una disposizione rettangolare di numeri:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \cdots & a_{3n} \\ a_{41} & a_{42} & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & a_{m5} & \cdots & a_{mm} \end{pmatrix}$$

Ogni numero a_{ik} (per es. a_{11} oppure a_{34}) rappresenta un elemento della matrice (gli indici degli elementi servono solo per differenziazioni, i nomi delle matrici sono, come noto, sempre lettere maiuscole, in questo caso A). La matrice è composta da righe (sequenza orizzontale di elementi) e colonne (sequenza verticale di elementi). Si dice che è del tipo (m,n), dal momento che possiede m righe ed n colonne. Si parla anche di una matrice (m,n). Gli elementi a_{11}, a_{12}, a_{13}, ... (oppure, più brevemente, a_{1k}) rappresentano la prima riga, tutti gli elementi a_{2k} la seconda riga ecc. Lo stesso accade con le colonne.

Un elemento a_{ik} si troverà quindi nella riga i e nella colonna k. In questo caso avremo a che vedere con una matrice bidimensionale.

Il numero di elementi in una riga può (ma non necessariamente) essere uguale al numero degli elementi in una colonna. In questo caso parleremo di matrice quadrata ad n righe, esempio:

$$A = \begin{pmatrix} 1 & 5 & -6 \\ -7 & 13 & 8 \\ -1 & 2 & 22 \end{pmatrix}$$

Questa matrice ha tre righe ed è quadrata.

Un altro caso particolare di matrici è costituito da quelle che possiedono una sola riga o una sola colonna. Di conseguenza esse saranno del tipo (1,n) oppure (m,1), (vedi sopra), esempio:

$$A = (1 \quad 5 \quad 3 \quad -4)$$

In questo caso si tratta di una matrice che possiede una sola riga (in questo caso con 4 "colonne"). Essa viene chiamata anche "vettore di riga". L'altro caso è il cosiddetto "vettore di colonna", esempio:

$$A = \begin{pmatrix} -4 \\ 9 \\ 6 \end{pmatrix}$$

Addizione/sottrazione di due matrici:
Una addizione oppure sottrazione può avere luogo solo con matrici dello stesso tipo (stesso numero di righe e di colonne).

L'addizione di due matrici C = A + B ha luogo elemento per elemento. Ogni elemento a_{ik} di una matrice verrà addizionato all'elemento corrispondente b_{ik} dell'altra matrice per ottenere l'elemento corrispondente c_{ik} della matrice di risultato. Vale:

$$c_{ik} = a_{ik} + b_{ik} \qquad \text{per tutti i,k}$$

Lo stesso vale anche per la sottrazione:

$$c_{ik} = a_{ik} - b_{ik} \qquad \text{per tutti i,k}$$

La moltiplicazione di una matrice per un numero:

Una moltiplicazione B = n*A di una matrice a piacere con un numero reale ha luogo anch'essa elemento per elemento. Ogni elemento a_{ik} della matrice viene moltiplicato per il numero n. Vale:

$$b_{ik} = n * a_{ik} \qquad \text{per tutti i,k}$$

Addizione, sottrazione e moltiplicazione per un numero soggiacciono alle stesse regole della operazioni normali con i numeri.

Moltiplicazione di matrici concatenate:
La moltiplicazione di due matrici è definita solo per matrici concatenate. Con il termine matrici concatenate intendiamo due matrici A e B, la prima delle quali possieda un numero di colonne uguale a quello di righe della seconda. Un esempio di una moltiplicazione permessa sarebbe quindi il seguente:

$$A * B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

A possiede tre colonne, B tre righe, per cui la condizione è soddisfatta. Vediamo anche tuttavia, che la moltiplicazione B*A non è permessa. Anche se B*A fosse permessa, ciò non significherebbe che il risultato sarebbe lo stesso di A*B. La sequenza dei fattori è infatti molto importante. Teniamo quindi presente che una moltiplicazione matriciale tra due matrici quadrate (vedi sopra) è sempre permessa.

Il risultato di una moltiplicazione matriciale è di nuovo una matrice. Essa possiede lo stesso numero di righe della prima (A) e lo stesso numero di colonne della seconda (B). Il risultato della moltiplicazione di cui sopra (matrice (3,3) per matrice (3,2)) sarebbe quindi una matrice del tipo (3,2) (3 righe, 2 colonne).

Il calcolo di questa matrice di risultato C è un po' complesso. Scriviamo, con abbreviazioni matematiche, il calcolo di un singolo elemento per i lettori esperti:

$$c_{ik} = \sum_{j=1}^{n} a_{kj} * b_{jk} = a_{il} * b_{1k} + a_{i2} * b_{2k} + \ldots + a_{in} * b_{nk}$$

cosa che significa: l'elemento cik viene determinato dalla somma di tutti gli $a_{ij} * b_{jk}$, con j=1 fino a j=n.

Dove:

c_{ik} un elemento della matrice di risultato C
a_{ij} un elemento della matrice A
b_{jk} un elemento della matrice B
n numero delle colonne di A, quindi anche: numero righe di B

Seguendo quanto dice la formula calcoliamo un elemento c_{ik} della matrice di risultato, moltiplicando tutti gli elementi della riga i di A con l'elemento corrispondente della colonna k di B e sommando risultati:

$$C = A * B$$

$$= \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$= \begin{pmatrix} 1*1 + 2*3 + 3*5 & 1*2 + 2*4 + 3*6 \\ 4*1 + 5*3 + 6*5 & 4*2 + 5*4 + 6*6 \\ 7*1 + 8*3 + 9*5 & 7*2 + 8*4 + 9*6 \end{pmatrix}$$

$$= \begin{pmatrix} 22 & 28 \\ 49 & 64 \\ 76 & 100 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{pmatrix}$$

Regole di calcolo:

A*(B*C) = (A*B)*C

Mentre sarebbe sbagliato: A*(B*C) = (A*C)*B

Come già determinato in precedenza, è importante ricordare che A*B = B*A NON E' VERO.

Esistono numerose matrici particolari, per le quali è possibile moltiplicare un'altra matrice, senza che cambi nulla. Si tratta di matrici quadrate, nelle quali tutti gli elementi sono uguali a zero. Solo gli elementi della cosiddetta diagonale principale devono possedere il valore 1, per questo vengono chiamate anche matrici quadrate unitarie E:

$$(1) \text{ oppure } \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ oppure } \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Vale:

A*E = A

8.1.5 Matrici di trasformazione

Nelle seguenti righe troviamo riassunte ancora una volta tutte le matrici di trasformazione affrontate nel presente libro. Tutte le matrici vengono indicate qui sotto forma di coordinate omogenee.

8.1.5.1 Matrici bidimensionali

Scala (Ingrandimento/Riduzione):

Matrice di scala:

$$S(S_x, S_y) = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Dove:

S_x fattore di scala in direzione x
S_y fattore di scala in direzione y

A seconda del valore di S_x o di S_y, l'immagine verrà modificata come segue:

	$S^x/_y > 1$	Ingrandimento
	$S^x/_y = 1$	Nessuna modifica
$0 <$	$S^x/_y < 1$	Riduzione
	$S^x/_y < 0$	Specchiatura contemporanea attorno all'asse x/y

Scala di un punto:

$$P' = P(x,y) * S(S_x,S_y)$$
$$= (x\ y\ n) * S(S_x,S_y)$$

Sotto forma di parametri:

$$x' = S_x * x$$
$$y' = S_y * y$$
$$(n' = n)$$

Traslazione (spostamento):

Matrice di traslazione:

$$T(T_x,T_y) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{pmatrix}$$

Dove:

T_x spostamento in direzione x
T_y spostamento in direzione y

Traslazione di un punto:

$$P' = P(x,y) * T(T_x,T_y)$$
$$= (x\ y\ n) * T(T_x,T_y)$$

Sotto forma di parametri:

$$x' = T_x * x$$
$$y' = T_y * y$$
$$(n' = n)$$

Specchiature:

Matrici

Specchiature attorno all'asse x:

$$M_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Specchiature attorno all'asse y:

$$M_y = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Sotto forma di parametri:
Specchiatura attorno all'asse x:

$$x' = x$$
$$y' = -y$$

Specchiatura attorno all'asse y:

$$x' = -x$$
$$y' = y$$

Rotazione attorno all'origine:

Matrice di rotazione:

$$R(w) = \begin{pmatrix} \cos(w) & \sin(w) & 0 \\ -\sin(w) & \cos(w) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Dove:

w = angolo di rotazione

Rotazione di un punto:

$$P' = P(x,y) * R(w)$$
$$= (x\ y\ n) * R(w)$$

Sotto forma di parametri:

$$x' = x*\cos(w) - y*\sin(w)$$
$$Y' = x*\sin(w) + y*\cos(w)$$
$$(n' = n)$$

Rotazione attorno ad un punto a piacere:

Matrice di rotazione:

$$R'(w,x_z,y_z) = T_1(-x_z,-y_z) * R(w) * T_2(x_z,y_z) =$$

$$\begin{pmatrix} \cos(w) & \sin(w) & 0 \\ -\sin(w) & \cos(w) & 0 \\ -x_z*\cos(w)+y_z*\sin(w)+x_z & -x_z*\sin(w)-y_z*\cos(w)+y_z & 1 \end{pmatrix}$$

Dove:

w angolo di rotazione
x_z centro di rotazione coordinata x
y_z centro di rotazione coordinata y

Rotazione di un punto:

$$P' = P(x,y) * R'(w,x_z,y_z)$$
$$= (x\ y\ n) * R'(w,x_z,y_z)$$

Sotto forma di parametri:

$$x' = x*\cos(w) - y*\sin(w) - x_z*\cos(w) + y_z*\sin(w) + x_z$$
$$y' = x*\sin(w) + y*\cos(w) - x_z*\sin(w) - y_z*\cos(w) + y_z$$
$$(n' = n)$$

8.1.5.2 Matrici tridimensionali
Scala (ingrandimento/riduzione):

Matrice di scala:

$$S(S_x,S_y,S_z) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dove:

S_x scala in direzione x
S_y scala in direzione y
S_z scala in direzione z

Scala di un punto:

$$P' = P(x,y,z) * S(S_x,S_y,S_z)$$
$$= (x\ y\ z\ n) * S(S_x,S_y,S_z)$$

Sotto forma di parametri:

$$\begin{aligned} x' &= S_x * x \\ y' &= S_y * y \\ z' &= S_z * z \\ (n' &= n) \end{aligned}$$

Traslazione (spostamento):

Matrice di traslazione:

$$T(T_x, T_y, T_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Dove:

T_x spostamento in direzione x
T_y spostamento in direzione y

Traslazione di un punto:

$$\begin{aligned} P' &= P(x,y,z) * T(T_x, T_y, T_z) \\ &= (x\ y\ z\ n) * T(T_x, T_y, T_z) \end{aligned}$$

Sotto forma di parametri:

$$\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \\ z' &= z + T_z \\ (n' &= n) \end{aligned}$$

Rotazioni attorno agli assi delle coordinate:

Rotazione attorno all'asse x:

Matrice di rotazione:

$$R_x(a) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & \sin(a) & 0 \\ 0 & -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dove:

a angolo di rotazione attorno all'asse x

Rotazione di un punto:

$$P' = P(x,y,z) * R_x(a)$$
$$= (x\ y\ z\ n) * R_x(a)$$

Sotto forma di parametri:

$x' = x$
$y' = y*\cos(a) - z*\sin(a)$
$z' = y*\sin(a) + z*\cos(a)$
$(n' = n)$

Rotazione attorno all'asse y:

Matrice di rotazione:

$$R_y(a) = \begin{pmatrix} \cos(a) & 0 & -\sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dove:

a angolo di rotazione attorno all'asse y

Rotazione di un punto:

$$P' = P(x,y,z) * R_y(a)$$
$$= (x\ y\ z\ n) * R_y(a)$$

Sotto forma di parametri:

$x' = x*\cos(a) + z*\sin(a)$
$y' = y$
$z' = -x*\sin(a) + z*\cos(a)$
$(n' = n)$

Rotazione attorno all'asse z:

Matrice di rotazione:

$$R_z(a) = \begin{pmatrix} \cos(a) & \sin(a) & 0 & 0 \\ -\sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dove:

a angolo di rotazione attorno all'asse z

Rotazione di un punto:

$$\begin{aligned} P' &= P(x,y,z) * R_z(a) \\ &= (x\ y\ z\ n) * R_z(a) \end{aligned}$$

Sotto forma di parametri:

$$\begin{aligned} x' &= x*\cos(a) - y*\sin(a) \\ y' &= x*\sin(a) + y*\cos(a) \\ z' &= z \\ (n' &= n) \end{aligned}$$

Rotazione attorno a tutti e tre gli assi, nella sequenza asse x, asse y, asse z:

Matrice di rotazione:

$$R_x(a)_y(b)_z(c) = R_x(a) * R_y(b) * R_z(c) =$$

$$\begin{pmatrix} \cos(b)\cos(c) & \cos(b)\sin(c) & -\sin(b) & 0 \\ \sin(a)\sin(b)\cos(c) - \cos(a)\sin(c) & \sin(a)\sin(b)\sin(c) + \cos(a)\cos(c) & \sin(a)\cos(b) & 0 \\ \cos(a)\sin(b)\cos(c) + \sin(a)\sin(c) & \cos(a)\sin(b)\sin(c) - \sin(a)\cos(c) & \cos(a)\cos(b) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dove:

a angolo di rotazione attorno all'asse x
b angolo di rotazione attorno all'asse y
c angolo di rotazione attorno all'asse z

Rotazione di un punto:

$$\begin{aligned} P' &= P(x,y,z) * R_x(a)_y(b)_z(c) \\ &= (x\ y\ z\ n) * R_x(a)_y(b)_z(c) \end{aligned}$$

Sotto forma di parametri:

$$\begin{aligned} x' &= x*A + y*B + z*C \\ y' &= x*D + y*E + z*F \\ z' &= x*G + y*H + z*I \\ (n' &= n) \end{aligned}$$

Dove:

$A = \cos(b) * \cos(c)$
$B = \cos(b) * \sin(c)$
$C = -\sin(b)$
$D = \sin(a) * \sin(b) * \cos(c) - \cos(a) * \sin(c)$
$E = \sin(a) * \sin(b) * \sin(c) + \cos(a) * \cos(c)$
$F = \sin(a) * \cos(b)$
$G = \cos(a) * \sin(b) * \cos(c) + \sin(a) * \sin(c)$
$H = \cos(a) * \sin(b) * \sin(c) - \sin(a) * \cos(c)$
$I = \cos(a) * \cos(b)$

Rotazioni attorno ad un asse a piacere:

Matrice di rotazione:

$R_{asse}(a) =$
$T(-x_0,-y_0,-z_0) * R_x(w_1) * R_y(w_2) * R_z(a) *$
$R_y(-w_2) * R_x(-w_1) * T(x_0,y_0,z_0)$

Con:

$$T(-x_0,-y_0,-z_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{pmatrix}$$

$$T(x_0,y_0,z_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{pmatrix}$$

$$R_x(w_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & z/c & y/c & 0 \\ 0 & -y/c & z/c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_x(-w_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & z/c & -y/c & 0 \\ 0 & y/c & z/c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(w_2) = \begin{pmatrix} c/d & 0 & -x/d & 0 \\ 0 & 1 & 0 & 0 \\ x/d & 0 & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(-w_2) = \begin{pmatrix} c/d & 0 & x/d & 0 \\ 0 & 1 & 0 & 0 \\ -x/d & 0 & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

dove:

a	angolo di rotazione attorno all'asse
x_0,y_0,z_0	coordinate di un punto sull'asse
x,y,z	coordinate di un altro punto dell'asse

e:

$$c_2 = y_2 + z_2$$
$$d_2 = x_2 + c_2$$

Rotazione di un punto:

$$P' = P(x,y,z) \quad * \; R_{asse}(a)$$
$$= (x\ y\ z\ n) \; * \; R_{asse}(a)$$

8.2 Funzioni di Library utilizzate nel presente libro

Nei diversi programmi di questo libro abbiamo fatto uso piuttosto spesso delle numerose funzioni di Library che ci vengono messe a disposizione dal sistema operativo dell'Amiga. Nei singoli capitoli esse sono state spiegate solo molto brevemente. Per questo motivo mostriamo qui in appendice un'altra spiegazione breve ma significativa.

Naturalmente si tratta di una parte molto esigua delle funzioni di Library effettivamente disponibili. Per ulteriori informazioni, si consiglia di esaminare la documentazione relativa al sistema operativo dell'Amiga.

Funzioni DOS/Exec:
CloseLibrary(Library)

La biblioteca precedentemente aperta con OpenLibrary( ) viene di nuovo chiusa.

Input:
Library Indirizzo della struttura di library

Output:

Tipi di dati in C:

```
void CloseLibrary();
z.B.: struct GfxBase *Library;
oder: struct IntuitionBase *Library;
```

Descrizione:

Quando non si ha più bisogno delle funzioni di una library per il programma, la si chiuderà con questo comando. Dopo di esso non sarà più possibile richiedere funzioni di library da tale library.

Exit(codice errore)

Termine di un programma (Funzione DOS).

Input:

Codice errore — Codice errore per programma da chiamarsi

Output:

Tipi di dati in C:

```
void Exit();
```

Descrizione:

Exit( ) termina il programma. La funzione non torna, di conseguenza, nel programma. Se il programma era stato avviato sotto CLI, il "codice errore" verrà interpretato come un codice di Return.

Message = GetMsg(Port)

Preleva il messaggio successivo dalla Message-Port.

Input:

Port — Indirizzo della porta di ricezione messaggio

Output:

Messaggio — Indirizzo del messaggio ricevuto oppure 0

Tipi di dati in C:

```
struct Message *GetMsg();
z.B.: struct IntuiMessage *Message;
```

Descrizione:

Con questa funzione (assimilabile ad una funzione di INPUT) riceveremo dati da qualunque apparecchiatura (esempio tastiera, mouse oppure disco ecc). La funzione, tuttavia, non aspetta un messaggio, nel caso in cui nessun messaggio sia presente. L'attesa

è impostabile per esempio con Wait( ). Se invece è presente un'informazione (esempio una pressione di tasto), il comando restituirà l'indirizzo della struttura di messaggio corrispondente, diversamente zero.

La struttura di messaggio in una finestra di Intuition sarà per esempio la seguente:

```
struct IntuiMessage
{
    struct    Message ExecMessage;
    ULONG     Class;          /* Flag IDCMP dell'evento        */
    USHORT    Code;           /* Numero menu, tasto ecc.       */
    USHORT    Qualifier;      /* Inform. addiz. SHIFT, Alt ecc*/
    APTR      IAdress;        /* Indirizzo di Gadget, Screen   */
    SHORT     MouseX, MouseY; /* Coordinate del mouse          */
    ULONG     Seconds, Micros; /* Ora di sistema dell'evento */
    struct    Window *IDCMPWindow; /* Indirizzo finestra       */
    struct    IntuiMessage *SpecialLink;
};
```

Tutti questi elementi di struttura non vengono utilizzati per ogni tipo di segnalazione. Da questa struttura sarà possibile prelevare quindi le informazioni necessarie sul tipo e contenuto del messaggio. Quando il messaggio non è più necessario, dovremo restituirlo con il comando **ReplyMsg( )**.

Library = OpenLibrary(LibNome, Versione)

Crea l'accesso ad una Library.

Input:

LibNome	Nome della Library
Versione	Numero versione della Library

Output:

Library	Indirizzo della struttura di Library oppure 0

Tipi di dati in C:

```
struct Library *OpenLibrary();
char *LibName;
long Version;
```

Descrizione:

La funzione OpenLibrary( ) permette l'accesso ad una Library. Una eventuale Library non residente in ROM dovrà venire caricata dal disco. Prima del caricamento, infatti, non sarà possibile gestire nessuna funzione che appartenga a tale library. Nel caso in cui l'accesso non dovesse essere possibile, la funzione fornirà uno 0. Prima della fine del programma sarà necessario richiudere la Library tramite il comando Close Library( ).

ReplyMsg(Messaggio)
Restituisce un messaggio.

Input:

Messaggio	Indirizzo del messaggio

Output:

Tipi di dati in C:

```
void ReplyMsg();
struct Message Message;
```

Descrizione:

Con questa funzione si restituisce il messaggio ricevuto (eventualmente con segnalazione).

Signal = Wait(maschera segnale)
Attende uno o più segnali.

Input:

Maschera segnale	Maschera del bit dei segnali che dovranno venire attesi (Bit = 1), oppure non attesi (Bit = 0).

Output:

Segnale	Numero del segnale che si è presentato (numero bit)

Tipi di dati in C:

```
long Wait();
long Signalmaske;
```

Descrizione:

Wait( ) attende uno o più segnali determinati. Tali segnali vengono riservati per esempio con il flag IDCMP. L'applicazione è ricavabile dai programmi Demo.

Funzioni di Intuition:
CloseScreen(Screen)

Chiusura di uno Screen di Intuition precedentemente aperto con OpenScreen( ).

Input:

Screen	Indirizzo della struttura di screen

Output:

Tipi di dati in C:

```
void CloseScreen();
struct Screen *Screen
```

Descrizione:

Con questa funzione è possibile chiudere uno screen sotto Intuition. In tale screen non potranno esserci delle finestre aperte. Dopo che è stato chiuso l'ultimo screen, questa funzione cerca di aprire lo screen di Workbench.

CloseWindow(Window)

Chiusura di una Window di Intuition aperta precedentemente con OpenWindow( ).

Input:

Window	Indirizzo della struttura di window

Output:

Tipi di dati in C:

```
void CloseWindow();
struct Window *Window;
```

Descrizione:

Con questa funzione è possibile chiudere una finestra (Window) sotto Intuition. Nella finestra non potrà più esserci nessuna riga di menu definita. Dopo la chiusura dell'ultima finestra in uno Screen di sistema (da non confondersi con uno screen Custom) questa funzione chiude contemporaneamente anche lo screen.

Screen = OpenScreen(NewScreen)

Apertura di uno screen di Intuition.

Input:

NewScreen	Indirizzo della struttura di NewScreen

Output:

Screen	Indirizzo della struttura di screen

Tipi di dati in C:

```
struct Screen *OpenScreen();
struct NewScreen *NewScreen;
```

Descrizione:

Si tratta della funzione centrale per l'apertura di un nuovo screen sotto Intuition. Tuttavia, prima di poter chiamare questa funzione, sarà necessario inizializzare una struttura chiamata NewScreen (in Basic dovremo inoltre riservare ed occupare uno spazio di memoria sufficiente). Questa struttura contiene dati importanti sul nuovo screen da approntare (esempio numero dei colori, risoluzione, ecc). Come valore di risposta otterremo l'indirizzo della struttura di screen vera e propria, nella quale potremo trovare i dati della struttura di NewScreen ad alcuni altri dati. La costituzione della struttura di NewScreen è presente nei programmi di esempio.

Window = OpenWindow(NewWindow)

Apertura di una Window di Intuition.

Input:

NewWindow — Indirizzo della struttura di NewWindow

Output:

Window — Indirizzo della struttura di Window

Tipi di dati in C:

```
struct Window *OpenWindow();
struct NewWindow *NewWindow;
```

Descrizione:

Questo comando apre una nuova finestra di Intuition. Nel caso in cui si sia dato come tipo di Screen CUSTOMSCREEN nella struttura di NewWindow, sarà necessario avere già aperto tale screen tramite OpenScreen( ). Se invece la finestra deve venire aperta in uno screen standard, tale screen verrà aperto automaticamente, nel caso in cui non esista ancora.

Prima però di poter chiamare questa funzione, sarà necessario inizializzare una struttura con il nome NewWindow (in Basic riservare ed occupare uno spazio di memoria corrispondente). Questa struttura contiene dati importanti sulla finestra da approntare (gadget, dimensioni, posizione, ecc). Come valore di risposta otterremo l'indirizzo della struttura di Window vera e propria, nella quale sarà possibile trovare i dati della struttura di NewWindow ed alcuni altri dati. La posizione della struttura di NewWindow è presente nei programmi esempio.

ScreenToBack(Screen)

Lo screen indicato viene spostato dietro tutti gli altri screen.

Input:

Screen — Indirizzo della struttura di screen

Output:

Tipi di dati in C:

```
void ScreenToBack();
struct Screen *Screen;
```

Descrizione:

Con questo comando si ottiene esattamente lo stesso risultato di quando si preme il gadget di profondità (DEPTH ARRANGEMENT) di uno schermo: lo screen in questione viene posto dietro tutti gli altri.

ScreenToFront(Screen)

Lo screen indicato viene posizionato davanti a tutti gli altri.

Input:

Screen	Indirizzo della struttura di screen

Output:

Tipi di dati in C:

```
void ScreenToFront();
struct Screen *Screen;
```

Descrizione:

Con questo comando si ottiene lo stesso risultato di quando si preme con il Mouse il gadget di posizionamento anteriore: lo screen in questione viene posizionato davanti a tutti gli altri screen.

Funzioni Grafiche:
ClearScreen(RastPort)

Cancellazione dell'intera Rasterport a partire dalla posizione attuale del cursore grafico.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge

Output:

Tipi di dati in C:

```
void ClearScreen();
struct RastPort *RastPort;
```

Descrizione:

Questa funzione cancella a partire dalla posizione attuale del cursore (posizionabile con Move( )) il resto della riga fino al bordo destro dello schermo. Quindi cancella anche tutto il resto del Raster fino al bordo inferiore. La cancellazione ha luogo con 0, mentre, in modo di disegno 2, con il colore impostabile tramite SetBPen( ).

Draw(RastPort,x,y)

Tracciatura di una linea dalla posizione attuale del cursore a x,y.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
x,y	Coordinate del punto finale

Output:

Tipi di dati in C:

```
void Draw();
struct RastPort *RastPort;
long x;
long y;
```

Descrizione:

La funzione presentata traccia una linea (colore da SetAPen) dalla posizione attuale del cursore grafico (regolabile con Move( ), oppure dall'ultimo punto tracciato), fino al punto avente le coordinate x,y. Questo punto finale della linea diventa anche il cursore grafico attuale.

DrawCircle(RastPort,mx,my,raggio)

Tracciato di un cerchio con il raggio Raggio e con il centro in mx, my.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
mx,my	Coordinate del centro del cerchio
raggio	Dimensione del raggio in pixel

Output:

Tipi di dati in C:

```
void DrawCircle();
struct RastPort *RastPort;
long mx;
long my;
long radius;
```

Descrizione:

Questa funzione traccia un cerchio (colore da SetAPen) con il raggio indicato e con le coordinate di centro mx, my.

DrawEllipse(RastPort,mx,my,raggio_x,raggio_y)

Tracciato di un'ellisse con i raggi raggio_x, raggio_y, e con il centro in mx, my.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
mx,my	Coordinate del centro del cerchio
raggio_x	Dimensione del raggio in direzione x (in Pixel)
raggio_y	Dimensione del raggio in direzione y (in Pixel)

Output:

Tipi di dati in C:

```
void DrawEllipse();
struct RastPort *RastPort;
long mx;
long my;
long x_radius;
long y_radius;
```

Descrizione:

La funzione presentata traccia un'ellisse (colore da SetAPen) con i raggi indicati e con le coordinate di centro mx, my.

Flood(RastPort,modo,x,y)

Riempimento di una superficie a piacere.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
x,y	Coordinate di un punto nella superficie

modo	Modo di riempimento: = 0: riempimento della superficie in questione, che possiede i colori del punto x,y. = 1: riempimento della superficie in questione che è circondata da punti che sono stati impostati con SetOPen( ).

Output:

Tipi di dati in C:

```
void Flood();
struct RastPort *RastPort;
long x;
long y;
long modus;
```

Descrizione:

Con questa funzione è possibile riempire a piacere delle superfici molto velocemente. A tale scopo indicheremo con x,y un punto all'interno di questa superficie. Il modo nel quale l'elaboratore riconoscerà i confini di una superficie, viene determinato con "modo". Nel caso in cui si voglia riempire solo la superficie che ha attualmente un determinato colore (modo = 0), cioè quello del punto con le coordinate x,y, la funzione disegnerà fino al punto nel quale incontra un'altro colore.

Nell'altro modo il riempimento avrà luogo finché la funzione non arriva ad un punto di bordo, che possiede il colore OUTLINE impostato con SetOPen. Questo è il modo più utilizzato.

Move(RastPort,x,y)

Spostamento del cursore grafico a x,y (nessun disegno).

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
x,y	Coordinate del nuovo cursore grafico

Output:

Tipi di dati in C:

```
void Move();
struct RastPort *RastPort;
long x;
long y;
```

Descrizione:

Sposta il cursore grafico interno non visibile (normalmente l'ultimo punto tracciato) al punto avente le coordinate x,y.

RectFill(RastPort,x1,y1,x2,y2)

Tracciatura di un rettangolo pieno.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
x1,y1	Coordinate del punto angolare superiore sinistro
x2,y2	Coordinate del punto angolare inferiore destro

Output:

Tipi di dati in C:

```
void RectFill();
struct RastPort *RastPort;
long x1;
long y1;
long x2;
long y2;
```

Descrizione:

Questa funzione traccia un rettangolo riempito con il colore impostato tramite SetAPen( ). La dimensione e la posizione del rettangolo vengono definiti tramite l'input di due punti angolari diametralmente opposti, cioè x1,y1 ed x2,y2.

SetAPen(RastPort,pal_reg)

Impostazione del colore di disegno attuale sul colore del registro di Palette pal⊇reg.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
pal_reg	Numero registro di Palette

Output:

Tipi di dati in C:

```
void SetAPen();
struct RastPort *RastPort;
long pal_reg;
```

Descrizione:

SetAPen( ) definisce un nuovo registro di Palette come fonte del colore attuale di disegno, tramite il quale verranno eseguite tutte le operazioni normali di disegno.

SetBPen(RastPort, pal_reg)

Impostazione del colore di sfondo attuale al colore del registro di Palette pal_reg.

Input:

RastPort	Indirizzo della RasterPort cui ci si rivolge
pal_reg	Numero registro di Palette

Output:

Tipi di dati in C:

```
void SetAPen();
struct RastPort *RastPort;
long pal_reg;
```

Descrizione:

SetBPen( ) definisce un nuovo registro di Palette come fonte del colore di sfondo attuale, che viene utilizzato da diversi comandi di disegno.

SetOPen(RastPort, pal_reg)

Impostazione del colore attuale di OUTLINE al colore del registro di Palette pal_reg.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
pal_reg	Numero registro Palette

Output:

Tipi di dati in C:

```
void SetOPen();
struct RastPort *RastPort;
long pal_reg;
```

Descrizione:

SetOPen( ) definisce un nuovo registro di Palette come fonte del colore attuale di OUTLINE, con il quale lavorano alcuni comandi di disegno. Contemporaneamente esso inserisce anche l'incorniciatura delle operazioni di riempimento di superfici. SetOPen( ) non è una funzione vera e propria, bensì una macro (definita con #define in

C, nel file di Include graphics/gfxmakros.h). In Basic, al contrario, dovremo utilizzare la seguente sequenza di comandi:

```
POKE WINDOW(8)+27,pal_reg 'Impostazione colore
flags = WINDOW(8)+32
POKEW flags, PEEKW(flags) OR 8 'Attivazione Flag di OUTLINE
```

SetDrMd(RastPort, modo)

Impostazione del modo di disegno attuale.

Input:

RastPort — Indirizzo della Rasterport cui ci si rivolge

modo — Modo di disegno: i seguenti valori determinano il modo di disegno e possono venire combinati tramite una addizione (operazione logica di OR):

= 0: (JAM1) — Il disegno viene effettuato solo con il colore di disegno (SetAPen( )). Se nello sfondo deve venire messo un punto, questo verrà ignorato. Un motivo di riempimento sarà quindi trasparente in quelle posizioni nelle quali il motivo contiene dei bit di zero.

= 1: (JAM2) — Vengono impostati sia i colori di disegno (SetAPen( )) che i colori dello sfondo (SetBPen( )). In questo modo, per esempio, un motivo di riempimento coprirà completamente tutta la grafica che si trova sotto di lui.

= 2: (COMPLEMENT) — Esegue una operazione logica di OR esclusivo (XOR) del numero di registro di Palette con i colori di tutti i punti cui ci si rivolge. Può trattarsi per esempio di tutti i punti che devono venire interessati solo dal colore del primo piano (combinazione JAM1 + COMPLEMENT).

= 4: (INVERSIVD) — I colori dello sfondo e del primo piano vengono scambiati l'uno con l'altro.

Output:

Tipi di dati in C:

```
void SetDrMd();
struct RastPort *RastPort;
long modus;
```

Descrizione:

Il comando imposta il modo di disegno desiderato, che verrà tenuto presente da tutte le funzioni di disegno.

SetRGB4(ViewPort,pal_reg,rot,gruen,blau)

Attribuisce ad un registro di Palette della ViewPort (Screen) una combinazione di colori.

Input:

ViewPort	Indirizzo della ViewPort cui ci si rivolge. Sotto Intuition la View-Port dello screen si trova nella struttura di screen. In C, si potrà quindi dare: &Screen -> ViewPort
pal_reg	Numero registro Palette
rot	Valore di intensità per quota di rosso
gruen	Valore di intensità per quota di verde
blau	Valore di intensità per quota di blu

Output:

Tipi di dati in C:

```
void SetRGB4();
struct ViewPort *ViewPort;
long pal_reg;
long rot;
long gruen;
long blau;
```

Descrizione:

Con questa funzione è possibile attribuire ad un registro di Palette una delle 4096 combinazioni di colore possibili.

error = Text(RastPort,string,anz_zei)

Output di una stringa di testo nel set di caratteri in vigore in quel momento.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
string	Indirizzo della stringa di testo da emettere
anz_zei	Numero dei caratteri nella stringa

Output:

error	= 0: nessun errore
	< > 0: errore

Tipi di dati in C:

```
BOOL Text();
struct RastPort *RastPort;
char *string;
long anz_zei;
```

Descrizione:

Text_ fa apparire un testo nella posizione attuale del cursore grafico.

WaitTOF_

Attesa finché il pannello elettronico non ha raggiunto il View successivo (Screen).

Input:

Output:

Tipi di dati in C:

```
void WaitTOF_;
```

Descrizione:

Con questa funzione abbiamo la possibilità di sincronizzare i nostri output grafici con il pannello elettronico del monitor. Infatti questa funzione attende finché il pannello elettronico non ha raggiunto la fine dello schermo. In questa maniera è possibile tracciare dei grafici senza sfarfallamento nel periodo impiegato dal pannello elettronico per raggiungere l'inizio dello schermo.

WritePixel(RastPort,x,y)

Impostazione di un punto a x,y.

Input:

RastPort	Indirizzo della Rasterport cui ci si rivolge
x,y	Coordinate del punto da posizionare

Output:

Tipi di dati in C:

```
void WritePixel();
struct RastPort *RastPort;
long x;
long y;
```

Descrizione:

Con questa funzione viene posizionato un singolo punto sullo schermo. Tale punto assumerà il colore di disegno attuale. Contemporaneamente tale punto diviene anche il cursore grafico attuale.

Finito di stampare nel mese di Maggio 1989
da Grafica '85 - Rodano Millepini (MI)

Altri libri di argomenti collegato al presente volume:

AUTORE	TITOLO	SUPPORTO	CODICE
Andrea Bigiarini Pierluigi Cecioni Marco Ottolini	**IL MANUALE DI AMIGA**		CZ532
Rita Bonelli Massimiliano Lunelli	**AMIGA 500 GUIDA PER L'UTENTE**		CC627
Peter Wollschlaeger	**AMIGA-ASSEMBLER**	DISCO 3½"	CL757
Edgar Huckert Frank Kremser	**AMIGA LINGUAGGIO C**	DISCO 3½"	CL758
Horst R. Henning	**AMIGA-BASIC**	DISCO 3½"	CL768

AMIGA
grafica 3D e animazione

Axel Plenge

Non sono solo gli appassionati di computer ad essere affascinati dalla grafica tridimensionale: chiunque resta sorpreso di fronte al sovrapporsi di effetti speciali assolutamente fantastici.
Al contrario, solo pochi sanno che non è affatto complicato programmare da soli tali effetti.
Nel presente libro è possibile apprendere la progettazione, programmazione e rappresentazione sul proprio Amiga di grafici tridimensionali professionali, iniziando proprio dalle problematiche più semplici.
Gli utenti Amiga appassionati di grafica, ma anche i programmatori esperti e i principianti nel settore della grafica, troveranno il testo, per la cui lettura sarà sufficiente una conoscenza minima della matematica del Basic e del C, chiaro e comprensibile.
Tutti i programmi presentati, esaurientemente commentati, sono contenuti nel dischetto allegato, e per essi è necessario, oltre all'Amiga 500/1000/2000, l'Amiga-Basic ed un compilatore C (Aztec C oppure Lattice C).

Sommario

- Elementi di base di grafica
- Operazioni bidimensionali
- Il mondo tridimensionale
- Linee e superfici nascoste
- Effetti di luce
- Riflessioni e ombreggiature
- Ray Tracing
- Colori
- Solidi di rotazione

GRUPPO EDITORIALE JACKSON

ISBN 88-7056-954-3



L. 59.000 Cod. CZ756